LEVEL I

(12)

ESD-TR-78-170 MTR-3531

MINICOMPUTER ARCHITECTURES FOR EFFECTIVE SECURITY KERNEL IMPLEMENTATIONS

BY JOHN D. TANGNEY

OCTOBER 1978

Prepared for

DEPUTY FOR TECHNICAL OPERATIONS
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
Hanscom Air Force Base, Massachusetts

Project No. 522N
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
Contract No. F19628-78-C-0001

REVIEW AND APPROVAL

This technical report has been reviewed and is approved for publication.

William R. Price

WILLIAM R. PRICE, Captain, USAF
Technology Applications Division

Charles J. Greene Jr.

CHARLES J. GREENE, Jr., Lt Colonel, USAF
Chief, Technology Applications Division

FOR THE COMMANDER

Eric B. Nelson

ERIC B. NELSON, Colonel, USAF
Acting Director
Computer Systems Engineering
Deputy for Technical Operations

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (18) (19) ESD TR-78-170	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) MINICOMPUTER ARCHITECTURES FOR EFFECTIVE SECURITY KERNEL IMPLEMENTATIONS,		5. TYPE OF REPORT & PERIOD COVERED
		6. PERFORMING ORG. REPORT NUMBER (14) MTR-3591
7. AUTHOR(s) (10) John D. Tangney		8. CONTRACT OR GRANT NUMBER(s) (15) F19628-78-C-0001
9. PERFORMING ORGANIZATION NAME AND ADDRESS The MITRE Corporation P.O. Box 208 Bedford, MA 01730		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Project No. 522N
11. CONTROLLING OFFICE NAME AND ADDRESS Deputy for Technical Operations Electronic Systems Division, AFSC Hanscom AFB, MA 01731		12. REPORT DATE (11) OCTOBER 1978
		13. NUMBER OF PAGES 86
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

(12) 88 p.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

COMPUTER SECURITY
SECURITY KERNEL
SECURE MINICOMPUTERS

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A security kernel is an isolated, protected, and highly reliable mechanism for providing information access controls within a computer system. Certain hardware features are important for an effective security kernel implementation. In this report the important architectural features are identified, and an evaluation of ... next page

(over)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

235 050

20. Abstract (continued)

cont.

commercially available minicomputer systems for their support of the important features is presented. The evaluation covers minicomputers offered by the following vendors: MODCOMP, DATA GENERAL, GENERAL AUTOMATION, VARIAN, PRIME, DIGITAL EQUIPMENT CORPORATION, HONEYWELL, IBM, INTERDATA, AND HEWLETT-PACKARD.

ACKNOWLEDGMENTS

This report has been prepared by The MITRE Corporation under Project No. 522N. The contract is sponsored by the Electronic Systems Division, Air Force Systems Command, Hanscom Air Force Base, Massachusetts.

Much of the work presented herein stems from the efforts of David Kallman, now working for ITT. David identified the minicomputer systems for evaluation, gathered appropriate literature from each vendor, and generated an initial evaluation of each machine. The author is responsible for the background material, refinement of the evaluation criteria, and the specifics and final form of presentation of the various evaluations. Of course, the author is solely responsible for any errors or omissions found in the final report.

The comments and suggestions of Ed Burke, Jerry Gann, and Jonathan Millen were especially valuable.

Eleanor Cashin deserves canonization for all her efforts and frustrations while serving between the author and a sometimes recalcitrant on-line document preparation system.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Concluded)

LIST OF TABLES

SECTION 1

INTRODUCTION

A technology has evolved to secure a general purpose computer utility* against the compromise and sabotage of information. This technology, called security kernel technology, is a disciplined approach to providing effective information access controls within a computer system.

A security kernel is an isolated, protected, and highly reliable component embedded within a computer's operating system. A combination of both hardware and software, a security kernel monitors and controls all accesses to information, permitting or denying access in accordance with a specific security policy.

While it may be theoretically possible to implement a security kernel on any machine, certain hardware architectural features are very important to implement effectively three essential characteristics of a security kernel, which are discussed later. As will be evident shortly, not all of the important hardware features are supplied as standard or optional features on all commercially available computers.

Smith [1] defined and described the important hardware features and also compared five large-scale, commercial, third generation computers to evaluate the suitability of each for an effective security kernel implementation. The five machines represented a broad range of architectural philosophies, but each possessed sufficient computational capacity to provide a general purpose computer utility. The Honeywell 6180 and Digital Equipment Corporation KI-10 were found to be the best candidates, with the 6180 providing just about all of the important features. The IBM 370 and Xerox Sigma 9 were judged to be difficult architectural bases for an effective security kernel implementation, while the Burroughs B6700 was considered an extremely poor choice.

* A general purpose computer utility is defined here as a multiprogramming, resource-sharing, computer system designed to support interactive and batch processing and to be accessible to multiple users concurrently.

Presently, many small to medium scale computer systems are marketed for use as general purpose computational utilities. This report will compare and evaluate against the important hardware architectural features, minicomputer systems offered by ten different vendors: MODCOMP, DATA GENERAL, GENERAL AUTOMATION, VARIAN, PRIME, DIGITAL EQUIPMENT CORPORATION, HONEYWELL, IBM, INTERDATA, and HEWLETT-PACKARD.

The remainder of this section will provide background information on security kernel technology. In Section II the specific hardware features that serve as the criteria for the evaluation are developed. The results of the evaluation are documented in Section III and a conclusion is presented in Section IV.

BACKGROUND

Information Security and Privacy

There are many commercial, industrial, military, and government environments with requirements for general purpose computer utilities - frequently minicomputer based - that permit the sharing of programs and data bases among users of the utility, while maintaining the security of information where essential and the privacy of information where desired.

The need to maintain the security of information is most visible in systems with requirements for multilevel secure information processing. Many military and government environments have experienced increasing demands to process concurrently information at multiple levels of classification on the same machine. If these demands are to be satisfied, effective information access controls must be provided to maintain the segregation of multilevel information and to insure that individuals cannot gain access to information classified above their level of clearance.

The need to maintain the privacy of information is evident in just about any computer utility. An owner of a program or data file - particularly a proprietary program or sensitive data file - must be confident that only (owner) designated individuals are permitted to access the file.

An effective security kernel implementation provides the required security and privacy controls. By creating an environment in which all accesses to information must occur through the kernel, the security and privacy of information can be guaranteed. By preventing unauthorized accesses to information, the kernel eliminates from the computer utility the threats of unauthorized modification, destruction, and purloining of information.

Security Kernel Technology

Since the late 1960's, the United States Air Force Electronic Systems Division (ESD) has sponsored research and development activities in the area of computer system security [2]. In 1972 ESD funded a Computer Security Technology Planning Study Panel to investigate and report on computer security issues, including the issue of insuring the security and privacy of information in a general purpose computer utility [3]. The panel's conclusion was that the operating system of a secure, general purpose computer utility should include an isolated, protected, and reliable mechanism known as a reference monitor. The reference monitor would guarantee the security and privacy of information by mediating and controlling all references (accesses) to information. A security kernel is the realization in hardware and software of the reference monitor concept.

The panel recognized that the effectiveness of the information access controls is a critical consideration. Clearly, it must not be possible to either circumvent or subvert the access controls. As the panel recognized, efforts to secure an operating system by either discovering and fixing all of its "holes", or by constructing a set of security features upon an existing, non-secure operating system, stood little chance at success. The former approach is ineffective because the absence of holes can never be demonstrated, only the inability to find them. The latter approach is ineffective because the security controls are built upon a non-secure operating system which can be penetrated, and the security controls at the outer level can be circumvented or subverted [4].

The only viable approach, the panel concluded, was to include security as a central consideration or goal during the design and development of an operating system. The design process should start with the formulation of a mathematical model of a secure computer system based on the reference monitor concept. Once a model is established and proved, a primitive, secure, machine that embodies the reference monitor concept is then designed, specified, and implemented. This primitive, secure machine consists of a base computer architecture - supplemented, of course, by hardware features important to an effective security kernel implementation - and a small amount of highly reliable security kernel software. Because of the requirement for high reliability, it must be possible to demonstrate or prove that the primitive, secure machine faithfully implements the mathematical model. Finally, with proof of the primitive, secure machine accomplished, an operating system is designed and implemented upon the secure machine. Since the operating system is itself constrained by the access controls provided by the primitive, secure machine, it is a secure operating system. Any software system constructed upon the primitive, secure machine and constrained by its access controls is a secure software system.

Mathematical Model

ESD sponsored several efforts [5, 6] to develop a mathematical model of a secure computer system based on the reference monitor concept. The specific security policy that the models were to enforce was the Department of Defense security policy of clearances and classifications.* In practice, clearances are assigned to all users of the computer system and classifications are attached to all information stored with the system.

One model, the Bell and LaPadula model [7], is the fundamental basis of all security kernel development activities at The MITRE Corporation. The Bell and LaPadula model defines a reference monitor based, secure computer system in terms of subjects, objects, and mathematical axioms that govern subject accesses to objects.

Subjects are information accessors, the active system elements such as users and processes operating on their behalf that read and write information.

Objects are system elements accessed by subjects; they are passive elements that serve as information containers. Examples of objects are program and data files which are stored in main memory and on peripheral storage media.

The reference monitor maintains the clearances of subjects and the classifications of objects, and it mediates and controls all

*Security levels are assigned to individuals (clearances) and information (classifications). A security level is composed of a linearly ordered classification level (i.e., Unclassified, Confidential, Secret, Top Secret) and a set of compartments (e.g., NATO, China, Nuclear).

subject accesses to objects in accordance with security policy. The policy consists of a non-discretionary security policy and a discretionary security policy.

Non-discretionary Security Policy

Non-discretionary security policy is defined by two model axioms: the simple security condition and the *-property.

The simple security condition governs a subject's read and execute accesses to objects. It states that a subject may read or execute an object if the clearance of the subject dominates* the classification of the subject. The simple security condition is drawn directly from DoD security policy; its intent is to prohibit users from obtaining information that they are not cleared to see.

The *-property governs subject write accesses to objects. It states that a subject may write into an object if the classification of the object is greater than or equal to the clearance of the subject. Stated another way, the *-property stipulates that a subject cannot write into an object classified at a level lower than the clearance of the subject. The *-property is designed to prevent a program operating on behalf of a user from reducing, accidentally or otherwise, e.g., via a "Trojan Horse" [8] the classification of information.

When an individual is granted a clearance, he is charged with responsibility for maintaining the classification of classified information. Normally, when the individual is working with pencil and paper, we can trust the tools he is working with not to compromise information, since the individual has very direct control over the pencil and paper. However, the tools a computer utility may provide cannot be similarly trusted and must be denied write access to objects of lower classifications. This is a consequence of: the limited and indirect control the individual has over the software operating on his behalf, the amount of information that may be compromised, the speed with which the compromise may occur, and the difficulty in detecting the violating program.

*A clearance is greater than or equal to a classification if: 1) the classification level of the clearance is greater than or equal to the classification level of the classification, and 2) the set of compartments of the clearance is a superset of the set of compartments of the classification.

The effect of *-property enforcement on a computer utility is to partition the information store into multiple levels of classification and to confine each information object to its associated partition, and thereby preventing the migration or leakage of information across partition boundaries.

Discretionary Security Policy

Discretionary security is DoD's need-to-know security policy; it states that, in addition to being properly cleared to access some particular piece of information, the user must have a valid justification for accessing the information. The implication is that the "owner" of a piece of information may decide to grant another individual access to the information, provided the individual is properly cleared and has the need-to-know.

In the model, then, in addition to maintaining the classification of all objects, the reference monitor maintains an access control list for each object. The access control list, which may be manipulated only by the object's owner, designates those subjects that have been granted access to the object and their permitted modes of access.

Thus, when a subject attempts to access an object, the reference monitor will verify that, for the requested mode of access, both non-discretionary and discretionary policies are satisfied.

Integrity

If the simple security condition, *-property, and access control lists are properly enforced by the reference monitor, classified information is protected from accidental or malicious disclosure to unauthorized users. However, as presented above, the model does not yet protect classified information against unauthorized modification or destruction. As formulated, the *-property permits a subject to write into an object classified at a level above the subject's clearance. There is nothing to prevent a subject from overwriting sensitive information that the subject cannot otherwise read. For example, the model permits a top secret process to execute an unclassified procedure, since execute access is equivalent to read access in the model. If the unclassified procedure is untrustworthy (i.e., coded by an uncleared individual), there is nothing to prevent it from destroying top secret information that is accessible to the process.

Biba has expanded the Bell and LaPadula model to address the problem of information sabotage [9]. In addition to attaching security levels to subjects and objects, the reference monitor also maintains an integrity level for all subjects and objects. A simple integrity condition and integrity *-property are defined that are the duality of the simple security condition and *-property. The integrity axioms prevent a subject of high integrity from accessing data or procedure objects of lower integrity. Low integrity procedures may only be used by subjects of equivalent integrity, and only data objects of equivalent integrity are accessible. The sabotage of high integrity information by low integrity procedures is therefore prevented.

Security Kernel Requirements

Three requirements for, or characteristics of, an effective security kernel implementation were identified by the panel. They are:

1. the complete mediation of all information accesses;

2. the isolation and protection of the security kernel from penetration and subversion; and

3. the consistent and reliable enforcement of the security policy.

The hardware features that constitute the evaluation criteria are derived directly from the three requirements. The features are introduced here in general terms and are more fully developed in Section II.

Complete Mediation

To state this requirement in the abstract terms of the mathematical model, the security kernel must intervene and mediate all accesses to objects, by subjects.

Objects have been defined as information repositories within a computer system, such as program and data files, and I/O storage devices. All objects reside on some type of memory, either main memory, secondary memory (disks and drums), or peripheral memory (magnetic tape, paper tape, punched cards, magnetic bubble memory).

Subjects have been defined as users and processes operating on behalf of users. The security kernel deals most directly with processes, which may be defined in any number of ways. A process is defined here as a collection of active objects (resources) and a process state. A process' address space is another term for the collection of active accessible objects; the address space includes all programs that may be executed, as well as all data files and I/O devices that may be read or written. The process state contains status information about the process, e.g., a security level, an integrity level, and an execution point. Since the definition of a process may also include a buffer area for interprocess communication messages, processes are objects as well as subjects.

I/O channels for direct memory access (DMA) I/O devices are another type of subject. I/O channels are small-scale processing units that execute I/O programs to move blocks of data between main memory and high-speed I/O devices. I/O channels are programmed (initialized) via commands from the central processor. As active system entities, the security kernel's access controls must also be applied to the operation of all I/O channels.

To restate the requirement for complete mediation of all subject-object accesses, a security kernel must mediate all:

1. process accesses to main memory;

2. process accesses to I/O storage devices;

3. I/O channel accesses to main memory;

4. I/O channel (and I/O device) accesses to processes (interrupts); and

5. process accesses to other processes.

The requirement for complete mediation of all process accesses to main memory implies that the security kernel must verify a process' access rights on all fetches and stores to main memory. Clearly, some additional circuitry must be present within the main memory addressing mechanism to check each and every access by the central processing unit to main memory. A virtual memory addressing system can provide this access checking circuitry. Virtual memory is a memory management scheme designed to support an environment for multiprogramming by partitioning total physical memory into distinctly accessible storage blocks. To support a multiprogramming environment where programs and data may be shared among processes, without one process destroying another, the virtual memory system will verify, at a minimum, read and write access rights on all accesses by the central processor to main memory.

The second, third, and fourth types of subject-object accesses above imply that the security kernel must be responsible for input/output access controls. A sufficient solution to insuring security kernel mediation of all I/O accesses is to permit only the kernel to perform I/O. Privileged I/O instructions are one means of restricting I/O capability to the kernel. Privileged instructions may be executed only when the central processor is operating in "privileged" mode. By permitting only kernel software to run in "privileged" mode, only the kernel can do I/O.

There are other ways of restricting I/O device access to security kernel software. Some architectures permit the central processor to address I/O device registers like main memory. Since the security kernel will maintain control over the virtual memory addressing mechanism, it can restrict the physical memory addresses of I/O device registers to its own address space. I/O device registers can be made inaccessible to user software.

Ideally, user software should be permitted to perform I/O under minimal security kernel control. This would reduce kernel responsibility for I/O, making the kernel smaller, less complex, and more easily verified. Hardware mechanisms that make user I/O possible are discussed in Section II.

The final type of subject-object access implies kernel mediation and control of interprocess communication. IPC control is part of the kernel's overall responsibility for control of system processes (creation, deletion, and switching of processes). Most process control functions will be implemented within the software portion of the security kernel. However, certain types of hardware features are identified in Section II that facilitate the kernel's process control responsibilities.

Isolation and Protection from Subversion

As the focal point of information access control, the security kernel must be protected from unauthorized modification and tampering that could render it ineffective.

Architectural support for the notion of execution domains* provides an environment in which the isolation and protection of security kernel components can be assured. The concept is well defined and dates back to the days of the early batch monitor systems when the importance of protecting supervisory programs and data bases from the vagaries of user software was first recognized. On those early systems two domains or modes of execution were typically provided: user and supervisor. A field within the central processor's status register indicated the current mode of execution. User software ran in user mode, which was unprivileged, while supervisory software ran in supervisor mode, which was privileged. Privilege meant that certain (privileged) instructions could be executed. From the perspective of protecting supervisory software, the important privileged instructions were those that manipulated memory protection features, since they could be used to protect memory areas where supervisory software resided. I/O instructions, as noted in the preceding subsection, were also privileged, so that the supervisor could control the movement of user programs and data into and out of main memory.

Security kernel software, of course, would run within privileged, supervisor execution domain. By properly managing the virtual memory system, the kernel would protect itself from user software confined to run within unprivileged execution domain.

The concept of a hardware ring structure, first described by Saltzer and Schroeder [11], is a particularization of execution domains to a virtual memory environment. Hardware rings extend the two domain arrangement into a multiple number of domains which are hierarchically structured or linearly ordered in terms of access rights to virtual memory objects. Conceptually hardware rings are arranged concentrically, with the innermost ring most privileged and the outermost ring least privileged. On a ring system the address

---

*It is important to note immediately that "domain" is used here in the sense of hardware provided "execution domains", following the terminology of much of the literature provided by computer manufacturers, and is distinguished from the notion of "protection domains", which is the subject of much research in computer security (see Reference 10) and is mostly provided in software through capability-based addressing.

space of a process consists of a number of virtual memory objects, each of which has been assigned to a specific ring (or rings) of execution. The address space is really dynamic, changing as the process' ring of execution changes. When the execution point of process moves into an inner ring, its address space increases as more objects become acessible and access rights to previously accessible objects increase. Newly accessible objects are those which have been assigned to the inner ring. Conversely the address space of a process decreases as its point of execution moves into an outer ring. As interring transfers, particularly inward ring transfers, are carefully controlled, objects assigned to inner rings can be protected from processes executing in outer rings.

From above, another form of privilege is the ability to execute privileged machine instructions. On ring systems privileged instructions are executable only by processes operating within the innermost, most privileged ring.

On ring systems security kernel software must execute within the innermost ring. Ring machines permit the implementation of a distributed kernel, which means that kernel procedures and data bases assigned to the innermost ring can be included within the address space of user processes. Not all kernel software would be so distributed, of course; only those procedures and data bases that should be accessible to user software would be distributed.

Correct Operation

The final security kernel requirement, the consistent and reliable enforcement of the security policy, is without doubt the most difficult to satisfy. The operation of the security kernel must be provably correct, so that the existence of implementation bugs that might be exploited by direct software attacks can be dismissed. This requirement for provably correct operation has motivated the development of a formal design, implementation, and verification methodology in security kernel technology.

The methodology employs a series of successively less abstract representations of a security kernel implementation, with a proof of correspondence between each representation and its preceding, more abstract representation [12].

Starting with the security policy to be enforced by the kernel, the first representation is a mathematical model of an abstract, secure computer system based on the reference monitor concept. The Bell and LaPadula model has been described above [7] . The next representation in the series is a formal top-level specification of the input/output behavior of the security kernel, using a specification technique originally proposed by Parnas [13] and extended by Price [14]. Millen has developed a technique to demonstrate that the top-level specification complies with the axioms of the mathematical model [15]. The next step is to implement the software portion of the kernel design on the proper hardware base, and to prove that the implementation corresponds to its top-level specification. Kallman and Millen have devised a kernel implementation and proof of correctness strategy [16], drawn from techniques developed at Stanford Research Institute [17], that involves the decomposition of the kernel into hierarchically structured levels of abstraction. The final step in the series is a useable secure machine in execution, obtained by compiling the kernel software developed in the preceding step. To complete the verification methodology, proof of a correct compilation must be demonstrated. Further research in this area of verification is needed.

The first two steps of the methodology, model and top-level specification, are independent of any particular hardware base. The third step is the implementation of kernel software on a proper hardware base. Hardware features that make a machine a proper base are those that contribute to a clean and efficient realization of the subject and object model abstractions. The result is a small, well organized, and understandable implementation, one that is receptive to complete testing and verification of correct performance.

A virtual memory organization is a desirable feature in this regard because it contributes to a homogeneous object structure where objects are accessed in a similar fashion. Complete homogeneity of object structure is supported by those architectures where I/O devices are also accessed via the virtual memory system.

The subject abstraction is supported by hardware features that permit a clean and efficient multiple process structure. Again, a virtual memory organization can assist here by providing a suitable environment for multiprogramming and by defining a per process address space in terms of object descriptors. Hardware assistance for a fast and efficient process switch and interprocess communication also contribute to an effective multiple process environment.

SECTION II

EVALUATION CRITERIA

The evaluation criteria are drawn directly from a set of architectural features previously identified by Burke [18] as contributing to an efficient and effective security kernel implementation on a minicomputer system.

Some of the features were introduced in the preceding discussion on security kernel requirements. Mention was made of the need to control processor accesses to main memory and I/O devices to satisfy the requirement for complete mediation. Virtual memory was identified as one means of managing and controlling accesses to main memory, and privileged I/O instructions were described as a sufficient means of restricting I/O capabilities to kernel software. Domains of execution and the more specific notion of concentric rings were noted as effective mechanisms for insuring the isolation and protection of kernel hardware and software. And finally, the requirement for security kernel verification demanded a clean and well structured kernel software implementation, facilitated by features that contribute to a good realization of subjects, via a robust multiple process environment, and objects, via a descriptor-based, virtual memory organization.

The criteria are presented under four functional areas, from above: virtual memory, I/O access control, execution domains, and multiple process control. In each area both essential and convenient features are developed. The essential features are just that - those features in each area that are absolutely necessary to satisfy the requirements for an effective security kernel implementation. Convenient features provide desirable capabilities in hardware, capabilities that would otherwise be provided by kernel software. Convenient features may contribute to the efficiency of a kernel implementation, to the verification of kernel software (by reducing its size and/or complexity), or to the support of a multilevel secure application system.

VIRTUAL MEMORY

Virtual memory is a somewhat overworked and frequently misused term, a term which to most individuals frequently connotes only the notion of a very large per process address space. The term is often applied to memory management systems that are really mapped memory systems. Most of the minicomputers that are evaluated support as an optional feature a memory mapping unit that permits the expansion of main memory beyond standard capacity. These machines are typically 16 bit machines that form 16-bit effective program addresses and support a per process address space of 64K bytes or words, depending upon the addressable unit. Even with the MMU and expanded main memory, the per process address space - now called a logical or virtual address space - remains at 64K. Expanded main memory is usually, but not necessarily, some multiple of 64K, and the MMU maps each process' virtual address space into a 64K physical address space in main memory.

Conversely, other machines surveyed (PRIME, SCOMP) form effective program addresses larger than 16 bits in length, something like 18, 20, or 22 bits. These machines provide a much larger per process virtual address space (e.g., 256M words on the PRIME machines), clearly much larger than the amount of physical main memory that is currently practical to attach. These machines more closely conform to the common notion of a virtual memory system.

The distinctions between mapped and virtual memory are not critical as far as the evaluation is concerned, other than the consideration that a very large per process virtual address space permits a very robust, multiple process, computational environment. The important characteristics of both mapped and virtual memory organizations are the following:

1) Total physical memory space is organized into and accessed in terms of logical storage units called either pages or segments. Pages are fixed length units, usually something like 512, 1,024, or 2,048 words/bytes in length. Segments are variable length units, often with some maximum length defined somewhere on the order of 32K or 64K words or bytes. Whereas segments are generally large enough to accommodate most programs and data files, several pages are required in a paged organization. Some systems provide a segmented-paged organization where each segment consists of an integral number of pages, up to some maximum number of pages. In a segmented-paged system, only the accessed page of a segment must be resident in main memory.

2) The page or segment corresponds to an object in the mathematical model. The virtual address space of a process is a collection of currently accessible objects, a set of active pages or segments. An object is made accessible (active) to a process when a descriptor for the object

is added to the table of active object descriptors for the process. The object is thereafter accessed (addressed) through its descriptor. The table of active object descriptors is a convenient definition of the process virtual address space.

3) Program effective addresses are virtual or logical addresses. They are two-component addresses, where the first component is a logical page or segment number - in effect an index into the active object descriptor table - and the second component is a word or byte offset location. The first component is used to locate the descriptor for the object within the process' active object table; among other items, the descriptor may contain the main memory location of the start of the object. The second component is added to the starting location to compute an effective physical address in main memory. If the addressed object is not currently resident in main memory, as detected by the memory mapping unit from a flag within the object's descriptor, a fault is generated during address translation and a (kernel) software routine is initiated to move the object into main memory from secondary (disc or drum) memory. A paged virtual memory organization is advantageous to memory management because only the accessed page of a program or data file must be moved into main memory.

4) Regarding the requirement for complete mediation, the most important characteristic is that, in most cases, all effective program addresses formed by a process are virtual addresses and must be translated into physical main memory addresses. During address translation, the mapping hardware verifies that the attempted mode of access is valid. The descriptor for the accessed object includes information defining the process' access rights to the page or segment. At a minimum the access control information will support write protection, where a bit within the descriptor must be set to permit write access to the object. Typically the mapping hardware will provide an unmapped mode of operation, when effective addresses are treated as physical, not virtual, addresses and access protection checks may be disabled. (The unmapped mode of operation would be restricted to kernel software). User software will run under mapped mode, so that all main memory accesses are mediated.

5) As for supporting the subject and object abstractions, the active object table is a succinct definition of the address space of a process. Descriptors are used to define the locations of objects as well as a process' access rights to them. Within a virtual memory environment, the active object table embodies most of the information that defines a process. (In a kernel based secure operating system, the table is manipulated by process management software modules within the kernel; objects are added to the table upon user process request, provided the simple security condition and *-property are satisfied.) When a process is allocated usage of the central processor, its table must be made accessible to the address translation hardware. For most mapped memory systems, this means that an image of the table is loaded from memory into a set of address translation or mapping registers in hardware or, often, in fast semi-conductor memory. For most virtual memory systems, this means that a single register is loaded that points to the table somewhere in main memory; since descriptors must be fetched from main memory for address translation, adding overhead to the time spent during address translation, a fast access cache is typically provided to hold frequently used descriptors.

On at least one virtual memory architecture (SCOMP), I/O devices are supported as part of the virtual environment. Descriptors are used to define I/O devices. A device is made accessible to a process when a descriptor for it is included within the process' active object table. The process accesses the device in terms of a virtual device address and the mapping hardware translates the access into an effective access to a physical I/O device. The result is a complete homogeneity of object structure in which all model objects are accessed in a similar fashion, i.e., by virtual addresses that are translated into physical memory addresses, or physical I/O device accesses, using information contained within process local object descriptors.

Essential Features

Paged or Segmented Virtual Memory

It must be clear that a virtual memory system is an essential architectural feature. Either a paged or a segmented organization will do. Both can provide an effective environment for multiple processes. Both require address mapping circuitry and can therefore provide access checking during address translation. Perhaps the ideal organization is a segmented-paged arrangement, with access rights applied at the segment level and segments partitioned into pages for efficient memory management.

Null, Read-Execute Total Access Rights

At a minimum null, read-execute, and total (e.g., read-write-execute) access permissions should be supported on a per page or per segment basis. The model axioms require the capability to segregate read access from write access. Null access means that no object is associated with the descriptor addressed by the first component of the virtual address; such an attempted access should generate an internal interrupt (or fault), as should all access violations detected by the mapping hardware.

Convenient Features

A finer grain of access permission is desirable. It would be convenient to be able to grant a process any subset of the following set of access rights: null, read, execute, and write. Lacking such fine granularity, read-only and execute-only assess permissions would be very helpful. Two other convenient features within virtual memory object descriptors, referenced and modified flags, are helpful to virtual memory management.

Read-Only

Read-only access permission is helpful because it can prevent a process from executing a data file. This is basically a validity check, but some applications have a requirement for no user programming and read-only permission makes this possible.

Execute-Only

Execute-only access is helpful in that proprietary programs can be protected. Programs stored in execute-only objects can be executed, but not read and copied.

Referenced Flag

A referenced flag is set within a virtual memory object descriptor when the object is accessed by a process. A referenced flag is helpful to memory management software when it must decide (e.g., least recently used algorithm) what object(s) in main memory can be replaced by other virtual memory objects.

Modified Flag

Similarly, a modified flag is set when a process writes into an object. An object that has been modified must be copied back to secondary memory when it is replaced in main memory, otherwise the modification is effectively lost.

I/O ACCESS CONTROL

A virtual memory addressing mechanism that mediates all process accesses to main memory is alone insufficient to satisfy fully the requirement for complete mediation. As information is also stored on and accessible from peripheral storage devices, an effective security kernel must control the movement of information to and from I/O devices.

All of the machines covered by this evaluation are based on a bus structured architecture where the functional modules of the minicomputer system - CPU, main memory, I/O devices - are attached to a common data and control path called a bus. The bus is the medium for intermodule communication. Briefly, one module communicates with another by first, gaining control of the bus; second, specifying the module to be communicated with by putting its address on the bus and waiting for acknowledgement; and third, putting the information to be transferred on the bus. Input/output occurs over the bus when data is transferred between an I/O device and either a CPU or main memory module.

Two very different forms of I/O must be considered, slow speed and high speed. Slow speed I/O is also called programmed I/O because the central processor is involved in the transfer of each and every word or byte of data. Slow speed data transfer results from the execution of I/O instructions by the central processor. The processor first executes an instruction to determine whether an I/O device is ready to perform I/O. The instruction includes the device's bus address and effectively reads the device's status register over the bus. If the device is ready, the processor will execute an instruction to read/write a word or byte of data rom/to the device. On a read, the I/O instruction sends a control word over the bus instructing the device to read a word or byte from its storage media and to put it on the bus for the central processor. On a write, the instruction puts a control word on the bus informing the device that it will be written on, followed by the data for the device. While the above is a very general description of programmed I/O on a bus structured architecture, it is meant to convey the notion that the central processor performs I/O by communicating with the status, control, and data registers of an I/O device over the bus. Programmed I/O is useful only with slow speed I/O devices like terminals, unit record devices, and paper tapes.

High speed I/O involves, naturally, high speed devices like magnetic discs and drums. High speed I/O involves only minimal central processor involvement, as information is transferred directly between the high speed storage media and main memory. High speed devices are also called direct memory access (DMA) devices; once initiated and started by the central processor, a DMA device works in parallel with the CPU and competes with it for access to main memory. Taking magnetic discs as an example, several disc drives may be controlled by a single disc control unit. Often a specialized processing unit called an I/O channel will connect several disc controllers to the bus. The central processor initializes (programs) the I/O channel for each DMA transfer. DMA transfers move data in blocks and initialization involves the specification of a particular device and other parameters for the transfer; e.g., location of the information on the device, starting address and length of block in main memory, and direction of transfer. The central processor initializes the transfer by passing control information over the bus to the I/O channel. After initialization the central processor starts the channel in operation and is no longer involved in the transfer until it receives an indication that the transfer is complete or that something has gone wrong.

Essential Feature

Access to I/O Devices is Controlled

It is essential that a minicomputer provides some mechanism that would enable a security kernel to maintain control over accesses to I/O devices. Clearly, unregulated access to multilevel peripheral storage devices by untrusted user or supervisory software cannot be permitted.

A sufficient solution is the notion of privileged I/O, which means that I/O is performed only when a process is executing in the privileged domain (i.e., the processor is operating in a privileged mode). Privileged I/O is implemented by privileged I/O instructions, which may be executed only when the processor is operating in privileged mode. Attempted execution of a privileged instruction in non-privileged mode results in an illegal instruction trap (internal interrupt) to a handler in privileged mode. The kernel, whose software will execute in privileged mode, must perform all I/O upon request by user or supervisory software.

Another means of implementing the notion of privileged I/O is possible. Whereas most machines employ 6 or 7 bit bus addresses (select codes) for I/O devices, some machines (e.g., PDP-11/45) have bus architectures where I/O device status, command, and data registers are addressed just like main memory. Fetching or storing into an I/O device register is no different than fetching or storing into any other main memory location. By judicious control over virtual to physical address translation, access to I/O device registers can be restricted to security kernel software.

Convenient Features

Although privileged mode I/O is sufficient to guarantee security kernel control over I/O device accesses, complete responsibility for I/O is delegated to the kernel, increasing kernel size and complexity at the expense of verification effort. It would be convenient if non-privileged user or supervisory software could perform some external I/O* under minimal kernel control. Two features are identified that contribute to this realization.

Mapped I/O Devices

An attractive approach is one, introduced earlier, that includes I/O devices as part of the descriptor-based virtual memory environment, where descriptors are provided for virtual I/O devices and the memory mapping unit translates virtual device accesses into physical device accesses. Consider, as an example, a bus architecture like the PDP-11 family of minicomputers where I/O device registers are addressed like main memory and descriptors can be included within a process' active object table that map into the control, status, and data registers of I/O devices.** A process would generate a kernel request for access to an I/O

*External I/O is distinguished from internal I/O. Internal I/O is swapping disk or drum I/O that the kernel must perform for virtual memory management. External I/O is what is commonly thought of as I/O, involving peripheral storage media like terminals, tapes, cards, etc.

**This is more easily said than accomplished, as will be evident later.

device; the kernel would grant or refuse the request depending on the security levels of the process and the device.* If the kernel grants the request, it enters a descriptor into the process' active object table.

This concept is acceptable only for programmed I/O devices, since these devices are only accessed by the currently executing process and data is transferred directly between (typically) a CPU register and the data register for the device. High-speed DMA I/O requires some additional control mechanism to check main memory accesses by the I/O channel, since the I/O channel may cause a security compromise, as well as destroy kernel code, if it does not follow its instructions, i.e., it reads or writes main memory locations it hasn't been instructed to access.

<u>DMA Main Memory Accesses Are Controlled</u>

To fully support the notion of mapped I/O devices - both programmed and DMA devices - all DMA device accesses to main memory must be mapped just as central processor accesses are. Now the virtual memory mapping unit assumes the role of general access controller for the minicomputer system.

A process would request access to a DMA device, just as above for a slow speed device; the kernel would grant a descriptor permitting the process to access the device's status and control registers. The process could then initialize a DMA block transfer using process local virtual addresses and start the data transfer. The DMA device would present virtual addresses which would be mapped into physical addresses using the process' active object table. The mapping unit would also check access rights. So that DMA transfers on behalf of several processes can occur concurrently, as virtual addresses are presented from a DMA device the mapping unit must be capable of associating the correct active object table for the process that initiated that device.

A serious drawback to this feature is that high speed I/O is now performed interpretively through the kernel's mapping unit. Some of the throughput of the high-speed transfer is traded off in favor of increased security and kernel software simplicity.

---

*The security levels of the process and device must be equal; hence the device may be read or written. Also, I/O devices cannot be shared among processes.

PROTECTION DOMAINS

Execution Domains

Most of the machines surveyed are execution domain machines. Several (PRIME, SCOMP) are concentric ring machines. The PDP-11 family is advertised as a three domain machine, but is very much like a concentric ring machine.

All of the execution domain machines provide a single privileged supervisor domain and one or more equally unprivileged user domains. The number of user domains is governed by the number of sets of address mapping registers within the memory mapping hardware. Some machines provide as few as one set, others as many as eight. The more sets provided, the greater the number of processes that can be supported with minimal process switching latency. Proper memory management by supervisor domain software can assure the protection of user domains from each other.

All of the domain machines provide privileged instructions, although a wide variation exists as far as what operations are privileged. Some examples of privileged instructions are those that perform I/O, manipulate virtual memory mapping registers, alter/load the processor status register (e.g., set the domain of execution), and enable/disable/mask interrupts. Correspondingly, software that does I/O, manages the virtual memory environment (including the handling of page or segment faults and access violations), manages processes, and handles interrupts - all functions that a security kernel must assume responsibility for - must run in supervisor domain.

Another form of privilege exhibited by execution domain machines is a capability for supervisor domain to operate in an unmapped mode, whereby effective program addresses are treated not as virtual addresses but rather as unmapped physical addresses. Typically access checking can also be disabled within supervisor domain. These two facilities can be used to give supervisor domain software unrestricted access to a block (usually the first 2**16 memory locations) of main memory. A security kernel running in supervisor domain would reserve this memory block for its own software and data bases, denying unprivileged user software access to the area by properly managing the memory mapping information.

The PDP-11 three domain architecture, as implemented by the memory management unit (MMU) option, closely approaches the concentric ring concept. In order of decreasing privilege, the three domains are: kernel, supervisor, and user. Associated with each domain is a set of memory mapping registers. A PDP-11 process consists of a kernel address space, a supervisor address space, and a user address space. When a PDP-11 process is executing in kernel domain, all three address spaces are accessible; when the process is executing in supervisor domain, both supervisor and user spaces are accessible; and when in user domain, only user space is accessible. Clearly, only kernel software would operate in kernel domain. On the PDP-11/45 a security kernel can be distributed among user processes.

Concentric Rings

Three of the machines evaluated, PRIME's 400 and 500 and Honeywell's SCOMP, are concentric ring machines.

The PRIME machines provide 3 rings while the SCOMP provides four.* Conceptually, the rings are arranged concentrically, with ring 0 innermost, most privileged, and most protected, and rings 1, 2, etc., peripheral to ring 0 and of decreasing privilege and protection. The processor status word includes a field that defines the current ring of execution. As described earlier, a process' active program and data files are assigned to specific rings. This assignment of a process' active objects to specific rings is effected by ring bracket information stored within the descriptor for each object. Generally there are several brackets within each descriptor that define the rings of execution from which the object may be read, written, and executed. For example, the SCOMP has three brackets, R1, R2, and R3, that are used in the following manner. A SCOMP process may write an object provided it has been granted write access to the object <u>and</u> its current ring of execution is between 0 and (including) R1; the process may read an object provided it has read access <u>and</u> its current ring of execution is between 0 and R2; the process may execute an object provided it has execute access <u>and</u> its current ring of execution is between R1 and R2 (and note that the ring of execution will not change); and finally, the process may call (with a special instruction) and execute an object, with a resulting change in the current ring of execution to a lower ring (an inward ring call), provided the process has execute access <u>and</u> its current ring of execution is between R2 and R3 (R3 must be greater than R2 within the object's descriptor). As should be evident, a process' access capabilities

*Note that SCOMP rings 0 and 1 are identical, effectively providing just 3 rings.

to objects within its address space tend to increase as the process' current ring of execution decreases or moves inward. Also, some instructions can be defined as privileged to processes operating within ring 0 and, possibly, ring 1.

Within a concentric ring architecture like the SCOMP, objects are implicitly assigned to a ring or set of rings through the assignment of ring bracket values within descriptors for the object. For example, assuming a kernel that will run in ring 0, read and write access to a kernel data object can be restricted to kernel procedures by the assignments R1 = 0 and R2 = 0 for the object's descriptor. A kernel procedure object can be included within the address space of a user process with the assignment of ring bracket values as follows: R1 = 0, R2 = 0, and R3 = 3. This means that the user process, when executing in rings 1, 2, or 3, may call the kernel procedure using the special call instruction, resulting in a change of the current ring of execution to ring 0. The kernel procedure and data objects above are both assigned to ring 0.

Essential Hardware Features

Two Hierarchically Structured Domains or Rings

It is minimally essential that a machine provide two domains where one domain is privileged and protected from the other. Security kernel software would run within the privileged domain, and untrusted user software would execute within the unprivileged domain. All of the machines surveyed meet this essential requirement.

Controlled Transfer into Privileged Domain

Transfer of execution into the privileged domain must certainly be controlled. Clearly, transfer to arbitrary points within a security kernel running on privileged domain cannot be allowed.

All of the machines surveyed meet this essential requirement. In general, transfer into privileged domain occurs as the result of external and internal interrupts.

External interrupts are controlled signals generated by I/O devices (i.e., external to the central processing element); internal interrupts are unexpected signals generated by hardware conditions or programming violations (i.e., typically within the central processing element). Examples of internal interrupts are memory management and protection faults and illegal instruction traps. External interrupts cause a transfer to a predefined entry point, dependent upon the interrupting device, which holds the location, generally within the privileged domain, of a software handler for the device. Memory management faults (e.g., addressed object not in core, memory protection fault) also cause automatic transfer to predefined locations in privileged domain space. Attempted execution of privileged or undefined instructions within an unprivileged domain also results in automatic transfer into privileged domain. On the execution domain machines, one such instruction is a supervisor call instruction which is used within unprivileged domain to request supervisory services from privileged domain software. As might be expected, ring architectures have a more sophisticated means of domain transfer, which is discussed shortly.

Convenient Hardware Features

Three or More Domains or Rings

Just having two domains is a constraint for providing protected supervisory services. Fundamental to kernel technology, due to the requirements for verification, is the need to minimize the amount of security sensitive software that must run in privileged domain. This consideration precludes the inclusion of supervisory services within kernel software and, within a two-domain machine, means that supervisory software must run in the same domain as and unprotected from user software. At least three domains are more convenient, so that a separate domain for supervisory software is provided.

Hierachically Structured Domains/Rings

If three domains/rings are provided, it would be convenient if they were arranged in a hierachical order of privilege and protection, so that a domain intermediate in privilege and protection is available for supervisory software. On most 3+ execution domain machines there are really only two levels of privilege, and a supervisor would be delegated to run in one of the unprivileged user domains. Although such a supervisor can be protected from software running in other user domains, a separate supervisor domain that is more privileged than user domain is more desirable. The PDP-11/45 and 11/70 provide a supervisor domain intermediate to an unprivileged user domain and a privileged kernel domain. On ring machines, the protection rings are ordered in terms of privilege and protection; a supervisor would run in ring 1, protected from, and more privileged than, supervisory and user software running in higher rings.

Multiple Privileged Domain/Ring Entry Points

It would be convenient if the trap instructions for user-software-initiated transfer into the privileged domain/ring provided more than a single entry point. With just one entry point, a kernel would have to retrieve additional information from user software to determine the particular kernel function requested; the kernel must then transfer to the function's real entry point. Multiple hardware-supported entry points would eliminate the overhead of determining the real entry point.

Most of the domain machines support only a single entry point. The ring machines provide a much more flexible mechanism for ring transfers. Recall that a transfer of execution into an inner ring can occur only by the invocation of a special procedure object (in SCOMP, R3 > R2) using a special procedure call instruction. The special procedure object is called a "gate" segment because a set of specific entry points are defined at segment creation. The address translation hardware will insure that, on an inward ring transfer to a gate segment, the effective transfer address is a legal entry point within the gate segment. Kernel gate procedures can be defined with several hardware-monitored entry points so that transfer into the kernel is tightly and completely controlled.

Argument Validation

When kernel functions are invoked, non-kernel software must often provide pointers for the passing of arguments or the return of results. These pointers must be validated; in other words, the kernel must determine that the user process really has access to the locations provided as pointers. For example, consider a user request for the kernel to input a block of information from an I/O device into an area within the user's virtual address space. The kernel must verify that the user process has write access to that area. Any hardware features that would minimize kernel software overhead for argument validation would be convenient.

On execution domain machines, a kernel should be permitted to use user domain mapping registers for address translation in fetching operands or storing results. Further, the kernel must be able to tolerate argument validation access faults; i.e., such access faults should set a flag or condition code accessible to the kernel, but should not generate an internal interrupt.

Argument validation is somewhat more complex in a ring machine because of the possibility of indirect addressing through different virtual memory objects in the course of effective operand address generation. That is, the processor may make one or more memory references to different objects to fetch indirect addresses before the operand is actually fetched. Indirect address fetch is subject to the same access control and address translation as a simple read access to data. For example, if an indirect address is fetched from an object that can be written from a higher ring than the current ring of execution, the ultimate location referenced by the indirect address must be governed by access controls defined by the ring in which the object containing the indirect address resides, rather than the current ring of execution. That is, the address translation and access checking hardware should validate indirect references with respect to the ring to which the object containing the indirect address has been assigned. The SCOMP, for example, accomplishes this by maintaining a register Reff, called the effective ring, that is the maximum (inclusive OR) of the R1 (write) ring bracket values in all descriptors encountered during effective address generation. Reff is initialized to the current ring of execution at the beginning of each instruction cycle and its value, as updated, is used as the effective ring of execution during all access checks for the duration of the cycle .

Support for Stacks

Most of the machines surveyed provide some hardware support for the notion of stacks. Stacks facilitate the implementation of shared, reentrant, and recursive procedures, and they provide an efficient mechanism for subroutine parameter passing and the return of results. Clearly, separate stacks must be maintained for the different domains/rings in which a process may execute. It would be convenient if some hardware support were provided for the establishment of appropriate stacks upon interdomain/interring transfers, and the reestablishment of old stacks upon return. Otherwise most of this stack management must be performed in software.

PROCESS CONTROL

Input/output control and storage control (virtual memory management) are two major activities of a security kernel. A third is the management and control of processes, the implementation of the mathematical model's subject abstraction.

Essential Hardware Features

Multiple Processes

If many users are to share concurrently the available resources of a general purpose computer system, the base computer architecture must provide support for an efficient multiple process structure. The minimal hardware support necessary is the capability to save and restore process definition information. Recall that a process is defined as an address space and a state (or context). The state of a process in execution is embodied by the current values of certain central processor registers, e.g., processor status (current domain, interrupt priority level, condition codes), stack pointers, program counters, general purpose and floating point registers. Within a virtual or mapped memory environment, the address space of a process is embodied by a set of address translation registers. To support multiple processes, a computer architecture must provide a means of saving and restoring the various hardware registers that define the state and address space of a process in execution.

Convenient Hardware Features

An Efficient Process Switch

Experience gained in the design of a multilevel secure application software system - a secure, interactive military message service designed to operate on a kernel-based secure operating system - has indicated that fast and efficient process switching may be critical to performance. The constraint on object modification imposed by *-property enforcement requires multiple processes (at various classification levels) operating on behalf of each message service user. Response time and overall system performance are critical factors governing user acceptance of a multilevel, secure, interactive military message service, and fast process switching contributes to better response and performance.

Any hardware support for minimizing the time required to save and restore process definition information is clearly convenient. For example, a single instruction to save/restore process state registers as a block is helpful, rather than repetitive instruction execution to save/restore a single register at a time. On execution domain machines, a single instruction to save/restore a set of address translation registers at a time is also desirable. Another factor on execution domain machines is the number of sets of address translated registers (i.e., the number of user domains) provided by the memory mapping hardware. Some provide as few as one, others as many as 8 or 16. The more provided, the greater the probability that a process switch will not require the swapping of a set of mapping registers. Ring machines do not have sets of mapping registers. Rather, segment descriptor tables are kept in main memory and descriptors are fetched during address translation.* To save/restore process address space information on a process switch, only a single register that points to the process' descriptor table in memory must be saved/restored.

Support for Interprocess Communication

Interprocess signalling and communication are essential activities within a computer system supporting multiple processes. These activities are integral to the scheduling, dispatching, and overall coordination of processes. On most machines, these functions are handled solely in software. Surely, any hardware support in this area is desirable in that a more efficient IPC mechanism contributes to more efficient management of multiple processes.

Summary

A tabular summary of the essential and convenient hardware features is presented below in Table 1. This summary serves as an evaluation criteria checklist and will be used in the next section to summarize the evaluation of each machine.

*To speed address translation, i.e., to minimize the frequency of descriptor fetches from main memory, the ring machines provide a high-speed, associative cache memory in which current (most active) descriptors are stored.

Table 1. Evaluation Criteria Summary

FUNCTIONAL AREA	ESSENTIAL FEATURES	CONVENIENCE FEATURES
Virtual Memory	Page or Segmented Virtual Memory Null, Read-Execute Total Access Rights	Read-only Access Request Execute-only Access Referenced Flag Modified Flag
I/O Access Control	Access to I/O devices is Controlled	User I/O DMA Device Access to Main Memory is Controlled
Execution Domains	Two Hierarchically Structured Domains or Rings Controlled Transfer into Privileged Domain	Three or more Domains/Rings Hierarchically Structured Multiple Entry Points Argument Validation Stack Support on Domain/Ring Crossing
Process Control	Multiple Processes	Efficient Process Switch Support for Inter-Process Synchronization/Communication

SECTION III

THE EVALUATIONS

In this section the minicomputers are evaluated. Again, the machines evaluated are those within a vendor's product line intended for use as general-purpose computational utilities. Typically, the machines chosen are those that can be configured with optional memory mapping units. There may be several such machines within a vendor's line, and in these cases the accompanying evaluation attempts to cover all of them.

Each evaluation first discusses the machine's support in each of the four functional areas: Virtual Memory, I/O Access Control, Execution Domains, and Process Control. Then, the machine is assigned a rating on its support for the various essential and convenient features. The rating is assigned as follows:

POOR (-)	The machine supports the feature not at all or very poorly.
GOOD (+)	The machine supports the feature to some extent, but not completely.
EXCELLENT (*)	The machine provides complete support for the feature.

MODCOMP IV/35

Modular Computer Systems, Inc., Ft. Lauderdale, Florida, offers three families of compatible minicomputer systems: MODCOMP I, MODCOMP II, and MODCOMP IV. The MODCOMP IV family, which consists of one model, the IV/35, is the only one that supports a memory management subsystem and is therefore the only one evaluated. The IV/35 is described [19] as a medium-to-large, multiprogrammable, 32-bit parallel, general purpose digital computer, specifically designed to be the host machine in real time, communication, and information processing computer networks.

Virtual Memory

The MODCOMP IV/35 includes a memory management system that provides a paged mapped memory organization. Memory pages are each

512 bytes in length. Main memory is expandable in 32K byte increments from 32K bytes up to 512K bytes.

There are eight RAM-implemented Address Mapping Files (AMFs) - numbered 0 through 7 - for use in virtual address translation. Each AMF virtual map may address up to 256 pages, so a per process address space of 64K words is provided. Program virtual addresses are 16 bits: 8 bits select a page within the process' AMF and 8 bits select a word within the page.

A process can have null, read, read-execute, or read-write-execute access to pages in its virtual address space. In addition, there are two map select registers that permit either one or two maps to be used concurrently for instructions and operands; therefore program and data files can be mapped separately, providing execute-only and read-only access.

Referenced and Modified flags are not supported, although the ZIMP, ZOMP, AMEN, DMEN privileged instructions are convenient features for memory management. ZIMP and ZOMP are used to clear blocks of contiguous registers within instruction and operand AMFs, respectively; ZIMP and ZOMP provide a flexible and efficient mechanism for zeroing part of an AMF for processes that do not use the whole virtual address space. AMEN and DMEN provide semi-automatic allocation and deallocation of physical memory pages.

<u>I/O Access Control</u>

The MODCOMP IV/35 includes a Primary Input/Output Processor (PIOP) that may provide both low-speed and high-speed I/O capabilities.

As a standard feature, the PIOP performs program controlled byte or word data transfers between general purpose CPU registers and device registers of up to 64 peripheral devices daisy-chained on a 16-bit wide party-line I/O bus. Privileged I/O instructions are provided for transferring data, commands, and device status codes to and from each device. Low-speed devices are not addressed like main memory, but rather are assigned select codes (numbers from 0 to 62 - on the I/O bus).

As an optional feature, the PIOP may support a Direct Memory Processor (DMP) for high speed data transfer with DMA devices interfaced to the standard I/O bus. The optional DMP provides 16 multiplexed, block transfer channels connected to memory through a separate memory port. A privileged I/O instruction (DMPI) is used to initialize the DMP for each block transfer. A limited measure of

DMA device control - in the form of virtual to physical address translation - is provided by the DMP. The DPI instruction takes as a parameter the address of an AMF immage stored in main memory. When in operation, a DMP channel uses the map image to check access to main memory. At initialization, the DMP channel is also given a virtual address that is the starting location for the block transfer and a word count. The block to be transferred may be considerably larger than a memory page. The DMP channel does not verify access rights on each word access; rather, access checking and virtual address translation is performed only when a virtual page boundary is crossed. It is the responsibility of the operating system not to change any of the pages while the DMP is operating on them.

Execution Domains

The MODCOMP IV/35 offers but two hierachically structured execution domains: privileged and user. Eight mapped user domains are provided by the 8 hardware AMFS; all 8 user domains are equally unprivileged. Transfer to the privileged domain can be initiated from each user domain via the Request Executive Service (REX) unprivileged instruction. REX is inflexible, trapping via an Unimplemented Instruction Interrupt signal to a single entry point defined by an interrupt vector in low memory.

The privileged domain is characterized by the ability to execute privileged instructions. Some of the privileged functions provided are:

1. Enter virtual adressing mode; when off, the CPU addresses main memory directly.

2. Monitor and control the priority interrupt system.

3. Execute program-controlled I/O instructions and initialize DMP I/O channels.

4. Move instruction and operand map images between AMFs and main memory.

Argument transfer between a user domain and the privileged domain is accomplished by the select operand map (SOOM) privileged instruction, which permits privileged software to use another AMF temporarily for operand fetches/stores. Access violations do not generate a memory protection fault and are effectively null operations. Privileged software can therefore not detect the occurrence of the access violation.

Two privileged instructions, Load Register from Memory (LDVM) and Store Register from Memory (STVM) provide an additional measure of data transfer and a reasonable amount of argument validation. These instructions permit the movement of a data word between a general purpose register and the virtual address space defined by an AMF map image in main memory. These are useful for transferring operands to/from the virtual address space of a process that is suspended and does not have its map image loaded within an AMF (e.g., interrupt driven I/O handlers). The load or store is permitted only if read-write-execute access rights are allowed to the process for the virtual page accesed; otherwise, the load or store is not permitted and the carry condition code is set. It would be much more helpful if a load was permitted provided read or read-execute access rights were present.

<u>Process Control</u>

The MODCOMP IV/35 can accommodate 7 user processes directly using seven of the eight AMFs - assuming one map is reserved for the supervisor. Considering a system supporting 7 users or less, process switching can be quite fast, involving just a switch of a 32-bit program status doubleward (PSD). The PSD contains such context as:

o program counter

o current selected instruction and operand AMF

o integer overflow history

o condition codes

o privilege state

o current selected general purpose register set

There are 16 general purpose register sets implemented in firmware. Hence, considering a system with 15 users or less, process switching can be limited to the time it takes to load an AMF with a map image from main memory using the LOMP or LDMP privileged instruction. These instructions consume approximately 1 microsecond per map entry, with a maximum of 256 microseconds for a full map. With more than 15 users, process switching may require the loading of both an AMF and one of the 16 general purpose register sets from main memory. The latter occurs via the MRBM and MMRB privileged instructions. It takes 5 microseconds to load a register set and 20 microseconds to both store and load.

There is no hardware support for interprocess communication.

Summary

Table 2 summarizes the evaluation of the MODCOMP IV/35. The MODCOMP IV/35 satisfies all of the essential hardware features. There is additional support for some of the more important convenient features - specific access rights, more than two execution domains, and argument validation - and the MODCOMP IV/35 is judged a fair to good candidate for a security kernel implementation.

PRIME

PRIME Computer, Inc., Framingham, Massachusetts, offers a family of plug-compatible central processors: PRIME 100, 200, 300, 400, and 500. PRIME machines are finding application in the areas of data communications, on-line data acquisition and control, transaction-based information processing systems, and multi-user computational utilities. The larger, more powerful PRIME 400 and PRIME 500 are the machines of interest here. Both support as a standard feature a segmented-paged virtual memory system and are designed for use as integrated interactive, queued-job, and real time systems.

Virtual Memory

The PRIME 400/500 provide a segmented-paged virtual memory organization. Page size is 1,024 words. Segment size may range from 0 to 65,536 words in increments of 1,024 words - 0 to 64 pages. Unpaged segments are not permitted. Physical main memory is expandable up to 8 M bytes in 64K byte modules.

The per process virtual address space is 512 M bytes, consisting of 4,096 segments. Thus, a process's active object (segment) table may contain 4,096 segment descriptors. Actually, each active segment table consists of four groups of 1,024 segments each. When a process is executing, four descriptor table address registers (DTARs) point to the main memory locations of the four groups of segment descriptors.

Access rights apply on a segment basis; they can be selected from among none, read, read-execute, read-write-execute, and gate.

Segment referenced and modified flags are hardware maintained for each page of a segment within the page map for each segment.

Table 2. MODCOMP IV/35

FUNCTIONAL AREA	ESSENTIAL FEATURES	RATING	CONVENIENCE FEATURES	RATING
Virtual Memory	Paged or Segmented Virtual Memory	*	Read-only Access	*
			Execute-only Access	*
	Null, Read-Execute Total Access Rights	*	Referenced Flag	-
			Modified Flag	-
I/O Access Control	Access to I/O devices is Controlled	*	User I/O	-
			DMA Device Access to Main Memory is Controlled	+
Execution Domains	Two Hierarchically Structured Domains or Rings	*	Three or more Domains/Rings	*
			Hierarchically Structured	-
	Controlled Transfer into Privileged Domain	*	Multiple Entry Points	-
			Argument Validation	+
			Stack Support on Domain/Ring Crossing	-
Process Control	Multiple Processes	*	Efficient Process Switch	+
			Support for Inter-Process Synchronization/Communication	-

RATING

* Excellent
+ Good
- Poor

I/O Access Control

All I/O instructions are privileged instructions on the PRIME 400/500.

There is also some hardware support for control of DMA devices. The mapped I/O feature permits a DMA device to access the entire physical memory (as large as 8 M bytes) although the I/O bus has only an 18-bit address width - for compatibility with PRIME 100, 200, and 300 processors. All virtual DMA addresses are translated into physical addresses using the page table for segment 0. For example, if a DMA device is to read from main memory, a page table is constructed and pointed to by a segment descriptor that occupies the first entry of the group of segment descriptors pointed to by DTAR0. The segment 0 descriptor will permit only read accesses to those pages defined by the page table. Hence, a block as large as 64 K bytes may be transferred under the PRIME 400/500's mapped I/O. If a security kernel were implemented, the kernel must maintain the necessary virtual-to-physical correspondences in effect for the duration of the DMA transfer.

Execution Domains

The PRIME 400/500 supports a concentric ring protection mechanism. The three rings - numbered 0, 1, and 3 - are hierarchically ordered. Ring 0 is the most privileged. A process operating in ring 0 may execute privileged instructions and has read-write-execute access to all segments in the system. Some of the functions provided by privileged instructions are:

1. loading the processor status register;

2. hardware support for Dijkstra's P and V semaphores in the form of notify and wait instructions; these instructions complement the PRIME 400/500's extensive hardware support for process switching;

3. input/output and priority interrupt control; and

4. modification of processor modes which, among other things, permits virtual address translation to be turned on and off.

Each segment descriptor includes two fields that define access rights to the segment. One field is set to define permitted access to the segment from procedures executing in ring 1, while the other defines permitted access from ring 3. In effect, then, a data segment is assigned to ring 0 exclusively by setting null access in the ring 1 and ring 3 fields of its descriptor; as a variation, read access could be specified in the ring 1 field, permitting ring 1 procedures to read the data segment. A procedure segment is assigned to ring 0 by setting, say, gate access in the ring 1 field and null access in the ring 3 field. Such a setting permits the procedure segment to be entered only from ring 1 via specific entry points defined within the ring 0 procedure.

Ring crossing is very flexible and occurs by means of the Procedure Call instruction (PCL). The PCL instruction:

1. computes the ring number of the called procedure;

2. allocates a stack frame for the called procedure;

3. saves the caller's critical state information in the new stack frame;

4. loads the critical state for the called procedure; and

5. evaluates the caller's argument pointers, storing a list of final effective addresses in the new stack frame.

PCL addresses an entry control block (ECB) within the procedure being called. The ECB contains the critical state information for the called procedure, such as a pointer (ring number, segment, offset) to the first executable instruction, the stack frame size to be allocated, the number of arguments expected and where in the new stack frame to put them, and central processor modes to be set.

The ring number of the called procedure depends upon the caller's access privileges to the segment containing the entry control block. No ring change occurs if the caller has read access. If the caller has gate access, the ring of execution is taken from the ECB pointer to the first executable instruction.

Following the PCL instruction in the calling procedure is a list of argument transfer templates which define the argument list. During execution of the PCL instruction, the list of templates is evaluated to generate a list of actual arguments (or pointers) in the new stack frame. As part of the evaluation, argument validation is performed. The called and calling ring numbers are ORed and the resulting "weakened" ring number is inserted into each argument (or pointer) transferred into the new stack frame. When the called procedure is running and references a memory location pointed to by an argument, it is granted only the weakened privileges as defined within the descriptor for the referenced segment. (This validation methodology is acceptable only if access faults can be tolerated in ring 0. Since the PRIME 400/500 has clean fault handling, including a stacking of critical machine conditions in a special ring 0 stack, access faults can be tolerated in most applications.)

When the inner ring procedure returns (via the PRTN instruction), its allocated stack frame is deallocated and the calling procedure's state is restored.

Process Control

The PRIME 400/500 are unique among all of the machines surveyed in their extensive hardware support for the subject abstraction.

The PRIME 400/500 make efficient memory usage of segment tables. As is usual with segmented-paged architectures, the active segment table and page tables are kept in memory, and an associative cache memory is used to speed up address translation. In most segmented-paged architectures, a single hardware register points to the active segment table. As described earlier in the section on virtual memory, the PRIME 400/500 have 4 descriptor table registers (DTARs) pointing to 4 groups of segment descriptors. The advantage of this arrangement is that the distribution of shared segments (e.g., kernel and supervisory) is more efficient. With a single active segment table, the table entries for shared segments are duplicated for all processes, wasting a considerable amount of memory. With the PRIME 400/500, the first two groups (2,048) of segment descriptors are shared among all processes and the respective DTARs need not be changed when processes are switched. The other pair of DTARs point to two groups of segments that are private to processes and must be stored and reloaded when processes are switched.

Process switching is quite fast and efficient. A combination of hardware and firmware automatically controls the allocation of the central processor to the highest priority process in a queue of processes ready for execution. There is no need for software intervention. Priority process scheduling and dispatching - including the saving and restoring of registers, and the allocation of the two hardware sets of registers - is implemented in microcode. (It is further claimed that six hardware register sets can be easily implemented.)

The PRIME 400/500 offers hardware support for Dijkstra's P and V semaphore operations. A semaphore defines an event whose meaning is shared among two or more processes. Associated with the semaphore may be a queue of processes awaiting the event; these processes are waiting, and are not on the ready list of processes ready for CPU allocation. A process signals an event by executing a NOTIFY instruction on the semaphore defining the event. As a result of the NOTIFY, a process on the waiting list for that event is moved onto the ready list of processes. When a process executes a WAIT instruction on a semaphore, it gives up the CPU and puts itself on the wait list of processes associated with the semaphore. As a result of the WAIT, the automatic process scheduling and dispatching microprogram is executed.

Summary

Table 3 summarizes the evaluation of the PRIME 400/500. Both are judged as excellent hardware bases for a security kernel implementation, as they satisfy all of the essential features and nearly all of the convenient features.

GENERAL AUTOMATION

Only one machine offered by General Automation, Inc., Anaheim, California, the GA-16/440, is evaluated. The GA-16/440 is advertised as a fast, powerful, and versatile minicomputer system for application in such areas as data communications, data acquisition and control, batch processing, process control and machine control [20]. When used in conjunction with the Memory Management System option, the GA-16/440 provides a multiprogramming environment for support of a multi-user computational utility.

Virtual Memory

The optional Memory Management System (MMS) provides a mapped memory organization with a logical per process address space of 64K words. Physical memory is expandable from 64K to 1,024K 16K modules. Although a maximum physical memory size of 1,024K words can accommodate 16 distinct logical address spaces, only 4 sets of mapping registers are provided in high-speed scratch-pad memory.

The MMS implements a paged virtual memory environment where each page is 1K words in size. Each of the four maps contains 64 mapping registers. Of the four maps, only three may be available for user processes, since map 1 may be employed for the mapping of DMA device accesses to main memory.

Table 3. PRIME

FUNCTIONAL AREA	ESSENTIAL FEATURES	RATING	CONVENIENCE FEATURES	RATING
Virtual Memory	Paged or Segmented Virtual Memory	*	Read-only Access	*
			Execute-only Access	+
	Null, Read-Execute Total Access Rights	*	Referenced Flag	*
			Modified Flag	*
I/O Access Control	Access to I/O devices is Controlled	*	User I/O	-
			DMA Device Access to Main Memory is Controlled	*
Execution Domains	Two Hierarchically Structured Domains or Rings	*	Three or more Domains/Rings	*
			Hierarchically Structured	*
	Controlled Transfer into Privileged Domain	*	Multiple Entry Points	*
			Argument Validation	+
			Stack Support on Domain/Ring Crossing	*
Process Control	Multiple Processes	*	Efficient Process Switch	*
			Support for Inter-Process Synchronization/Communication	*

RATING

*	Excellent
+	Good
-	Poor

There are write protect and execute protect bits in each page descriptor; in effect, these bits implement null, read, read-write, and read-write-execute access capabilities on a per page basis.

There is also some support for segmentation. The maps in scratch-pad memory can be loaded and stored in groups of 8 contiguous page descriptors, permitting easy implementation of 8K word segments.

I/O Access Control

The central element of the input/output system is the I/O bus; up to 64 peripheral devices may be daisy-chained on the I/O bus. Each device is assigned a unique 6-bit select code. All DMA devices on the bus are attached to a Multiplexed High Speed Data Channel Controller (MHSDC) that does all the bookkeeping and interface protocol for high-speed data channel operations.

With the MMS option, all I/O is performed within supervisor domain by means of privileged I/O instructions. Most I/O instructions are privileged; ones that are not are those that access the Floating Point Processor and the Arithmetic Unit, both of which are treated as I/O devices.

As noted, the MMS permits the use of a register map for DMA devices; all DMA device operations may be mapped through map 1 by the MHSDC. This arrangement permits a DMA block transfer to occur for a user that is not the currently executing user. Although the Arithmetic Unit is treated as a DMA device, its accesses to main memory are not mapped through map 1, but through the current user's map.

Execution Domains

The MMS implements an operating environment external to the CPU that consists of four domains: 3 mapped user domains using maps 0, 2, and 3; and an unmapped supervisor domain. This external operating environment is created by interfacing the MMS to the I/O bus as a DMA device (select code '39'). The MMS is activated and controlled by sequences of programmed I/O commands.

The MMS operates in either mapped or transparent mode. In transparent mode, the low 64K words of physical memory are addressed directly. The supervisor domain operates in transparent mode. The MMS may also operate in a privileged instruction detection mode; this mode is turned off when privileged supervisor domain is entered.

Transfer into supervisor domain by user domain software is accomplished by execution of the Service Call (SVC) instruction. SVC generates a non-inhibitable (NI) internal interrupt, forcing transition into the transparent mode and initiating execution at a predefined location within low core of supervisor space. A MMS status word will hold the address of the SVC instruction that may be used to recover an argument list or address to specify further the desired service. The argument list or address can be obtained by using the single cycle MMS instructions, which allows the privileged supervisor to access data words through any map. These instructions can be used for argument validation because an MMS fault can be tolerated within supervisor domain.

Process Control

The MMS option provides the environment for multiple processes. There is some support for an efficient process switch.

There are two sets of general purpose registers, each containing 16 registers implemented in a "scratch-pad" memory - 8 foreground and 8 background registers. At any point in time, either the foreground registers are accessible or the background registers are accessible, but not both. A register set can be loaded from, or stored into, main memory by the instructions, LARS and SARS, respectively. It takes about 15 microseconds to store and then load a register set.

Given just three register maps for user processes, a process switch may require the storing of an active map in memory and the loading of a map image for another process. There are MMS instructions to load and unload register maps. Map images are moved between MMS maps and main memory by direct memory access. All transfers are to/from supervisor space using real memory addresses, since the MMS does not translate its own DMA addresses. It takes approximately one microsecond per page descriptor loaded or unloaded.

There is no hardware support for interprocess communication.

Summary

The evaluation of the GA-16/440 is summarized in Table 4. While the GA-16/440 meets all of the essential criteria, it is rated as just a fair to good candidate for a kernel implementation because it lacks a good share of the convenient features.

Table 4. General Automation - 16/440

FUNCTIONAL AREA	ESSENTIAL FEATURES	RATING	CONVENIENCE FEATURES	RATING
Virtual Memory	Paged or Segmented Virtual Memory	*	Read-only Access	*
			Execute-only Access	-
	Null, Read-Execute Total Access Rights	*	Referenced Flag	*
			Modified Flag	*
I/O Access Control	Access to I/O devices is Controlled	*	User I/O	-
			DMA Device Access to Main Memory is Controlled	*
Protect-on Domains	Two Hierarchically Structured Domains or Rings	*	Three or more Domains/Rings	*
			Hierarchically Structured	-
	Controlled Transfer into Privileged Domain	*	Multiple Entry Points	-
			Argument Validation	+
			Stack Support on Domain/Ring Crossing	-
Process Control	Multiple Processes	*	Efficient Process Switch	+
			Support for Inter-Process Synchronization/Communication	-

RATING

* Excellent
+ Good
- Poor

IBM SERIES 1/MODEL-5

Just one IBM machine, the Series 1/Model 5 minicomputer, is evaluated [2?].

Virtual Memory

The optional Storage Address Relocation Translator Feature implements a paged virtual memory environment. Each page is 2K bytes in length and main memory may range from 16 K bytes to 128K bytes in 16K-byte increments. Eight sets of page descriptor registers are provided; each set consists of 32 registers. One set of registers is implicitly assigned to the privileged supervisory domain for the handling of interrupts.

There are two access control bits per page descriptor, a valid bit and a read only bit. When the valid bit is 0, the register cannot be used for translation. In combination, the two access control bits effectively provide null, read, and read-write access rights.

A process may use 3 register maps concurrently, one for all instruction fetches and two for operand fetches. Since instructions and data can be mapped separately, execute only and read only access rights are effectively implemented. Instruction space can only be executed, while operand space cannot be executed.

Referenced and modified flags are not provided in page descriptors.

I/O Access Control

A single I/O channel directs the flow of information between I/O devices and the central processor or main storage. As many as 256 I/O devices - addressable by an 8-bit select code - can be attached to the channel.

All I/O operations initiated by the processor occur via a single, privileged I/O instruction (IO). The effective address generated by IO points to a two-word immediate device control block (IDCB). The IDCB holds an I/O comand (interpreted by the I/O channel), the addressed device, and a one-word immediate data word. For programed I/O commands that write to a device, the immediate data word holds a data word for transfer; upon completion of programmed I/O commands that read a device, the immediate data word holds the data word read. For commands that initiate DMA device transfers, the immediate data word holds the address of a DMA device control block (DCB). All IDCBs and DCBs are in supervisor space.

There is some support for mapped DMA device access to main memory. A field within the DCB selects one of the eight sets of page descriptor registers for use in address translation during the DMA transfer. However, the read-only bit is disabled during address translation on DMA device accesses to main memory; as a result, write protection violations are ignored. Only the validity bit is checked; if the validity bit is not set, the logical page is invalid and an invalid storage address interrupt is generated.

<u>Execution Domains</u>

Eight domains are provided: one privileged supervisor domain and seven equally-unprivileged user domains. The seven user domains are defined by the seven register maps available for user processes; remember, map 0 is reserved for use by the supervisor.

There are privileged instructions that are executable only in the supervisor domain. These instructions permit the supervisor to:

1. perform I/O and enable/disable priority interrupt levels;

2. load and store register maps;

3. disable/enable the virtual memory translator; when disabled, physical memory is directly addressed; and

4. disable/enable the standard lock-and-key memory protection mechanism which, when enabled, is used when the virtual memory translator is disabled. By disabling the lock and key mechanism, the supervisor can give itself access to all of main memory.

Control is passed to supervisor domain by the occurrence of a variety of interrupts, both class interrupts and I/O device interrupts. One type of class interrupt is generated by the execution of the supervisor call (SVC) instruction in user domain. SVC causes transfer of execution to a fixed location in supervisor space defined by an interrupt vector also in supervisor space.

Arguments and results can be transferred between user and supervisor domains easily. The supervisor can execute privileged instructions that load/store the Address Key register (AKR), which defines the two active operand register maps and the single instruction-fetch register map. To fetch an argument or store a result, the supervisor simply loads the appropriate values into the AKR and performs the operand fetch or store. However, since memory write protection violations are ignored in supervisor domain, only a limited form of argument validation is provided. The supervisor cannot determine whether a user process really has write access to the virtual addresses provided, only that the referenced addresses are within the process' virtual space.

Process Control

There are four priority interrupt levels at which the central processor may operate. Associated with each level is a set of registers that may have to be stored/loaded during a process switch. Each set includes an Address Key Register, a group of 8 general purpose registers, a Level Status Register (LSR), which contains status information and condition codes for processing at that level, and a program counter for that level. Execution of the Set Level Status Block (SELB) privileged instruction may cause the processor to change priority interrupt levels. It may also require that the register set associated with the target level be loaded with a Level Status Block (LSB) from main memory. If so, the SELB instruction loads a LSB from main memory beginning with the location specified by the effective address.

If the process switch also requires the swapping of one or more of the seven user register maps, the additional overhead can be considerable because the Set Segmentation Register (SESR) privileged instruction loads only one register at a time.

There is no hardware support for interprocess communication.

Summary

Table 5 summarizes the evaluation of the IBM Series 1/Model 5 minicomputer.

Although all of the essential criteria are satisfied, it is only rated a fair candidate for a kernel implementation because roughly half of the convenience features are not supported.

VARIAN 70 SERIES

This evaluation will consider several models of the Varian 70 series of computers offered by Varian Data Machines, Irvine, California [22, 23]. The Varian 70 series was designed for maximum performance in instrumentation, data acquisition, and communications

Table 5. IBM Series 1/Model 5

FUNCTIONAL AREA	ESSENTIAL FEATURES	RATING	CONVENIENCE FEATURES	RATING
Virtual Memory	Paged or Segmented Virtual Memory	*	Read-only Access	*
			Execute-only Access	*
	Null, Read-Execute Total Access Rights	*	Referenced Flag	-
			Modified Flag	-
I/O Access Control	Access to I/O devices is Controlled	*	User I/O	-
			DMA Device Access to Main Memory is Controlled	+
Execution Domains	Two Hierarchically Structured Domains or Rings	*	Three or more Domains/Rings	*
			Hierarchically Structured	-
	Controlled Transfer into Privileged Domain	*	Multiple Entry Points	-
			Argument Validation	-
			Stack Support on Domain/Ring Crossing	-
Process Control	Multiple Processes	*	Efficient Process Switch	+
			Support for Inter-Process Synchronization/Communication	-

RATING

*	Excellent
+	Good
-	Poor

systems applications; with the MEGAMAP option, several models can be adopted for multiprogramming applications. Microprogrammed control stores permit a common instruction set among all machines in the series.

There are two families of interest within the Varian 70 series: V76 and V77. The basic differences between the two families is in the types of power supplies used and the fact that the V76 family is compatible with certain DMA devices used with Varian's earlier 620 minicomputer.

The models of interest are V76-1100, V76-1101, V77-400, and V77-600; the MEGAMAP virtual memory system is an optional feature on all of the four machines.

Virtual Memory

With MEGAMAP, physical memory may range from 32K words to 1,024K words of semiconductor RAM memory. MEGAMAP provides a 512-word paged organization with a logical, per-process address space of 32K words. There are 16 maps for virtual address translation also implemented in RAM memory; each map consists of 64 page descriptor registers. Note that the 16 maps can address a total of only 512K words, whereas 1,024K words of physical memory may be available.

There are four modes of access to each page: null, read, read-execute, and read-write-execute. Support for execute-only access is not provided.

Referenced and modified flags were provided in page descriptors by the memory management option for the V72, V73, and V74 families of the series. The memory management option that preceded MEGAMAP could address only 256K words. Unfortunately, to permit MEGAMAP to address 1,024K words, the reference and modified bits were stolen for use as physical address bits within the page descriptor.

I/O Access Control

With the MEGAMAP option, I/O instructions are treated as privileged operations. A memory protection interrupt will result from the attempted execution of an I/O instruction fetched from some map other than MAP 0. Since instruction fetches are drawn from MAP 0 only when MEGAMAP is operating in executive mode, I/O instructions are privileged to executive mode.

Three types of I/O are supported: programmed I/O, DMA, and Priority Memory Access (PMA).

Programmed I/O uses separate program instructions to transfer a byte or word to low-speed I/O devices and to initialize and start the operation of high-speed DMA and PMA devices.

DMA devices share a memory port with the central processor and steal memory cycles from the central processor to transfer blocks of data at high rates. Note that MEGAMAP is itself a DMA device; special memory map I/O instructions are defined to load and store MEGAMAP's control registers, thereby setting its operating modes, determining its status, and initializing and starting DMA transfers of page descriptor map images between main memory and MEGAMAP's RAM store. All DMA device transfers are mapped using a map specified during DMA initialization. All MEGAMAP DMA transfers are mapped via MAP 0.

The memory systems used in the V76 and V77 families are dual port memory systems. One port is shared by the central processor and DMA devices; the other port may be used by an optional PMA controller for very high speed data transfers between main memory and PMA devices - without interrupting the central processor. PMA controllers are initialized with programmed I/O instructions in a fashion similar to DMA controllers. All PMA transfers are mapped through MEGAMAP, using a map specified at PMA device initialization.

Execution Domains

MEGAMAP provides 16 domains: 15 user domains (via MAP 1 through MAP 15) and one privileged, executive domain (MAP 0 or unmapped direct access). There are only two hierarchically structured domains of privilege: user and executive.

MEGAMAP actually has three modes of operation: inactive, executive, and user. In inactive mode, address mapping is disabled and the first 32K words of main memory are accessed directly. All instruction fetches, operand fetches, and operand stores are unmapped. All instructions, including I/O instructions, may be executed. Inactive mode can only be entered from (and can be considered an extension of) executive mode.

In executive mode, address translation is enabled and all instruction fetches are via MAP 0. There are four states within executive mode; the maps used for operand fetches and stores depend on the particular state, as indicated below where MAP n refers to the active map indicator register.

STATE	INSTRUCTION FETCH	OPERAND FETCH	OPERAND STORE
0	MAP 0	MAP 0	MAP 0
1	MAP 0	MAP 0	MAP n
2	MAP 0	MAP n	MAP 0
3	MAP 0	MAP n	MAP n

By changing its state (via programmed I/O instructions), the executive domain can effect the transfer of data between any user domain and executive domain. Care must be exercised here though. In all states, several instructions (LDA1, LDB1, or LDX1) always fetch their operands using MAP 0. Also, to ensure that all instruction fetches are via MAP 0, indirect addressing must not exceed the first level in states 2 and 3, because after the first level of indirect addressing, instruction fetches in some cases are treated as operand fetches by the memory map.

Normally, memory protection checks are disabled when MEGAMAP is operating in executive mode. Hence, the above facility is useful for argument validation only when the checks are enabled in executive mode.

All instructions except HALT are permitted in executive mode. Inactive mode can be entered only from executive mode. Executive mode is entered from user mode by the occurrence of memory protection interrupts generated by CPU, DMA, or PMA accesses to main memory, or by the execution of any illegal instruction (I/O or otherwise) in user mode. Transfer is to specific, pre-defined locations in low core. There is a single entry into executive mode as a result of illegal instruction execution.

<u>Process Control</u>

MEGAMAP provides a suitable environment for multiple processes. Since there are as many as 15 user maps, most process switching should be limited to the swapping of central processor registers. If the number of user processes to be supported exceeds 15, a process switch may involve the storing and loading of a user map. MEGAMAP can load itself using DMA at the rate of approximately 1.5 microseconds per page descriptor. Any number of consecutive maps can be loaded or stored in one DMA block transfer.

Although all Varian 70 series machines are microprogrammable, there do not appear to be any instructions provided to optimize the swapping of the central processor's general purpose registers.

There is but one set of eight registers; each register must be stored/loaded separately.

There is no hardware support for interprocess communication.

Summary

Table 6 summarizes the evaluation of the four models of interest in the Varian 70 series. Since all of the essential hardware features and many of convenience features are provided, the V76-1100, V76-1101, V77-400, and V77-600 are all rated as good candidates for a security kernel implementation.

Considering other families within the Varian 70 series, the V72, V73, V74, and V75 families can all be judged as good candidates for a kernel implementation, provided the optional memory mapping subsystem that preceded MEGAMAP is implemented. While the earlier subsystem is limited to 256K of physical memory, the referenced and modified flag convenience features were supported. Other than these two flags and the physical memory capacities, the only other difference between the two mapping subsystems is that the earlier version supported either a 32K or 64K logical address space. With a 32K word logical space, up to 16 translation maps are possible; with a 64K logical space, up to 8 translation maps are possible. Otherwise, the two mapping subsystems are equivalent.

INTERDATA

Just one machine offered by Interdata, Inc., Oceanport, New Jersey, the 8/32, is evaluated [24]. The Interdata 8/32 is a high performance 32-bit minicomputer designed for use in process control, data communications, and multi-user time-sharing applications.

Virtual Memory

The Memory Access Controller (MAC) is a standard feature on the 8/32; it provides automatic program relocation and protection under a segmented virtual memory organization. Segments are variable in length, ranging from 256 bytes to 64 bytes in blocks of 256 bytes. Paging is not supported - an entire segment must be resident in main memory. Physical memory may range from 128K bytes to 1,024K bytes in increments of 128K bytes.

There is but one set of hardware mapping registers and it consists of 16 segmentation registers. Thus, a single user domain of 16 program and data segments is provided. The permitted modes of

Table 6. Varian 70 Series

FUNCTIONAL AREA	ESSENTIAL FEATURES	RATING	CONVENIENCE FEATURES	RATING
Virtual Memory	Paged or Segmented Virtual Memory	*	Read-only Access	*
			Execute-only Access	-
	Null, Read-Execute Total Access Rights	*	Referenced Flag	*
			Modified Flag	*
I/O Access Control	Access to I/O devices is Controlled	*	User I/O	-
			DMA Device Access to Main Memory is Controlled	+
Execution Domains	Two Hierarchically Structured Domains or Rings	*	Three or more Domains/Rings	*
			Hierarchically Structured	-
	Controlled Transfer into Privileged Domain	*	Multiple Entry Points	-
			Argument Validation	+
			Stack Support on Domain/Ring Crossing	-
Process Control	Multiple Processes	*	Efficient Process Switch	+
			Support for Inter-Process Synchronization/Communication	-

RATING

*	Excellent
+	Good
-	Poor

access supported on a segment basis are: null, read, read-write, read-write-execute, and read-execute. Neither referenced nor modified flags are provided within the segment descriptor, although a modified bit can be implemented with a minimal amount of software overhead.

I/O Access Control

All I/O instructions are privileged instructions that can only be executed when the central processor is in supervisory domain.

There is no mediation of DMA device operations. All DMA device accesses to main memory via the selector channel are absolute and unmapped.

Execution Domains

Only two execution domains are provided: one user domain in which program addresses are mapped using the 16 segmentation registers, and one supervisor domain in which program addresses are unmapped (absolute) and privileged instructions may be executed.

Transfer into supervisor domain is automatic upon the occurrence of an external or internal interrupt; execution of the Service Call (SVC) instruction in user domain initiates an internal interrupt with transfer to one of 16 entry points in supervisor domain. The entry point selected is dependent upon an operand to SVC supplied by the user.

A second operand to SVC is usually a pointer to the memory location of the arguments needed by supervisor domain software to complete the function specified. Argument validation can be effected by execution of the Load Real Address (LRA) instruction which simulates the operation of the MAC. Of all the domain-oriented machines surveyed, the LRA instruction of the Interdata 8/32 provides the best means of argument validation. In effect, LRA takes the virtual address pointer supplied as an operand to SVC and translates it into a physical address using a translation map image in main memory. (Note that LRA does not use the active map present in the segmentation registers, but rather a map image.) If a memory protection violation is detected, instead of a fault being generated, as would result during normal address translation by the MAC, the condition codes are set to reflect the type of violation detected. The argument validation mechanism provided in most of the other domain-oriented machines permits the supervisor to assume the access privileges of the user domain by using the user's address translation map to fetch the arguments. This scheme is only partially satisfactory because an access fault interrupt may occur and such faults are not always tolerable within the supervisor domain.

Process Control

The MAC provides an environment for the support of multiple processes.

There is some support for an efficient process switch in the form of multiple sets - as many as 8 - of general purpose registers. Four sets may be dedicated to interrupt handling at the four levels of interrupt priority, leaving four sets that may be shared among user processes and minimizing the need for storing and reloading a register set on a process switch. The Load Multiple (LM) and Store Multiple (STM) instructions can be used to load/store the currently active register set from/into consecutive memory locations.

Unfortunately, only one user domain of 16 segmentation registers is provided by the MAC. Switching user processes requires the storing and reloading of the segmentation registers - a major inconvenience. Values are loaded into the segmentation registers by storing into assigned memory locations in low main memory, with the MAC operating in unmapped mode. MAC registers are stored by reading the dedicated locations and storing the values elsewhere in main memory. To load or store MAC registers efficiently, the LM and STM instructions should be used, using a general purpose register set as intermediate storage.

There is no hardware support for interprocess communication.

Summary

The evaluation is summarized in Table 7.

The Interdata 8/32 meets all of the essential features, but due to lack of support for the convenience features in the areas of I/O Access Control and Virtual Memory, and considering the lack of a third domain, it can only be rated a fair candidate for a security kernel implementation

Somewhat less of a fair rating is be extended to the earlier 7/32 model. The 7/32 includes the MAC as an optional feature, whereas the MAC is part of the basic processor on the 8/32. Further, the 7/32 provides only 2 sets of general purpose registers (no priority interrupt levels) which, on the average, would tend to increase the overhead incurred on process switching.

Table 7. INTERDATA 8/32

FUNCTIONAL AREA	ESSENTIAL FEATURES	RATING	CONVENIENCE FEATURES	RATING
Virtual Memory	Paged or Segmented Virtual Memory	*	Read-only Access	-
			Execute-only Access	-
	Null, Read-Execute Total Access Rights	*	Referenced Flag	-
			Modified Flag	+
I/O Access Control	Access to I/O devices is Controlled	*	User I/O	-
			DMA Device Access to Main Memory is Controlled	-
Execution Domains	Two Hierarchically Structured Domains or Rings	*	Three or more Domains/Rings	-
			Hierarchically Structured	-
	Controlled Transfer into Privileged Domain	*	Multiple Entry Points	+
			Argument Validation	*
			Stack Support on Domain/Ring Crossing	-
Process Control	Multiple Processes	*	Efficient Process Switch	+
			Support for Inter-Process Synchronization/Communication	-

RATING

*	Excellent
+	Good
-	Poor

HONEYWELL'S SCOMP

One of the development activities sponsored by ESD in the area of secure computer systems was the design, implementation, and verification of a security kernel for Honeywell's Multics system [25], a large-scale, general purpose computer utility. Part of that effort was the development of a kernel-based secure front-end processor (SFEP) for secure Multics [26]. The hardware base for the SFEP development is Honeywell's SCOMP, a Level 6 Model 43 minicomputer [27] enhanced by a hardware Security Protection Module (SPM) that facilitates the conversion of the commercial, unprotected 6/43 into a kernel-based, secure front-end or communications processor. However, the SCOMP possesses sufficient computational capacity for application as a general purpose computer utility.

The SPM was specifically designed to serve as the hardware component of the SFEP security kernel. So it should not be surprising to find that it includes virtually all of the hardware features considered essential or convenient to an effective security kernel implementation. The function of the SPM is to mediate, through a descriptor structure, all interactions between the various modules (processors, I/O devices, memories) connected to the 6/43's MEGABUS. The SPM may be thought of as a general address translation and access mediator for a number of requestors - the modules connected to the MEGABUS.

Virtual Memory

The SPM implements a segmented-paged virtual memory organization, with a per process virtual address space of up to 64K words and a physical memory expandable to 1M words. A hardware register, the descriptor base register (DBR), points to a table of descriptors defining the objects (resources) - both virtual memory segments and I/O devices - accessible to the currently executing process.

Both paged and unpaged virtual memory segments are allowed. If unpaged, the segment descriptor points to the segment's physical address; if paged, the segment descriptor points to a page table for the segment.

Referenced and modified bits are maintained within both the segment descriptor and all page descriptors for paged segments. For unpaged segments they are maintained within the segment descriptor.

Access control information is maintained within the segment descriptor. It consists of three ring bracket fields (R1, R2, R3) and three 1-bit access permission fields (R, W, E). The access modes supported are any subset of the set (read, write, execute), constrained by the effective ring number, Reff, and the values of certain ring bracket fields. The calculation of Reff is described below in the subsection on Execution Domains; in general, the value of Reff is greater than or equal to the current ring of execution. The access control information is interpreted according to the following rules, where Reff, R1, R2, and R3 may assume the integer values 0, 1, 2, 3:

1) Write permission if (W = ON) and (Reff $\leq$ R1); rings 0 through R1 inclusive are defined to be the write bracket for the segment: processes executing in rings 0 through R1 may write the segment if (W = ON).

2) Read permission if (R = ON) and (Reff $\leq$ R2); rings 0 through R2 inclusive are the read bracket.

3) Execute permission if (E= ON) and (R1 $\leq$ Reff $\leq$ R2); rings R1 through R2 inclusive are the execute bracket.

4) If (E = ON) and (R3 > R2) and (R2 < Reff $\leq$ R3) execution will cause an inward ring transfer provided the CALL instruction is used; rings R1 through R3 are the call bracket; R2 becomes the new ring of execution; a segment defined by a segment descriptor where R3 > R2 is called a gate segment.

I/O Access Controls

I/O devices are supported within the virtual memory organization. They are accessed by process-local virtual device addresses, which are translated by the SPM into physical device addresses using a set of descriptors for I/O devices. The DBR points to both the set of I/O device descriptors and segment descriptors accessible to the currently executing process. Unlike segments, however, I/O devices cannot be shared among processes, so I/O device descriptors are not shared.

Access control information is maintained within the I/O device descriptor. It consists of the R1, R2, and R3 ring bracket fields and the R, W, and E access permission fields. The value of Reff is calculated by the SPM in the same manner as for memory accesses. The permitted modes of access are determined by the following rules:

1) Read permission (initiate a read from the device) if (R = ON) and (Reff $\leq$ R2); rings 0 through R2 are the read bracket.

2) Write permission (initiate a write to the device) if (W = ON) and (Reff $\leq$ R1); rings 0 through R1 are the write bracket.

3) Device control operations (status, positioning requests) are permitted if (E = ON) and (Reff $\leq$ R3); rings 0 through R3 are the control bracket.

There are two forms of DMA device control; DMA devices may be mapped or pre-mapped. The pre-mapped facility is provided so that the full performance capability of DMA devices can be attained without the overhead incurred by the SPM's virtual to physical memory address translation. The memory access checks are made by the SPM when the block transfer is initiated by the user process. The process supplies a virtual address for the block transfer and an extent (length). The SPM verifies that the device has been assigned to the process, that all memory addresses affected by the transfer have the proper access permission for the effective ring number and access mode of the process, that the affected memory addresses are described by a single direct memory page descriptor or unpaged segment descriptor, and that the process' I/O device descriptor allows the requested access mode for the process' effective ring of execution. If the SPM checks are verified, the SPM will give the DMA device a physical memory address and extent, and start the transfer. Because it is necessary to trust the operation of the I/O channel, certification of this type of I/O is not believed to be possible; this type of I/O was not included in the top-level specification of the SFEP security kernel [26].

With the mapped facility, DMA devices use virtual memory addresses that are translated and access checked by the SPM. When a mapped DMA device is initialized, the SPM verifies that the device is assigned to the process, and that the I/O descriptor for the device permits the requested mode of access for the effective ring of execution of the process. If the checks pass, the SPM initiates the block transfer. Since DMA I/O is asynchronous (i.e., a DMA transfer may have been initialized by a process different from the process currently executing on the processor), the SPM must remember, for each active mapped DMA device, the effective ring of execution and a set of memory segment descriptors for the process that initiated the I/O transfer.

Execution Domains

The SPM implements a concentric four-ring structure, where the rings are numbered 0, 1, 2, and 3, with ring 0 most privileged and ring 3 least privileged.

Ring crossing is controlled by the SPM, but is very flexible. Transfer to a gate segment assigned to an inner ring, from a process executing within the call bracket of the gate segment, is via the CALL instruction. The current ring of execution is changed to the value of R2 in the descriptor for the gate segment only if $R2 < Reff \leq R3$; otherwise the current ring of execution is unchanged because the gate segment has been called from within its execute bracket. The gate segment descriptor also defines a call limiter which defines a series of valid entry points at lower segment offsets. The SPM verifies that the second component (offset) of the CALL instruction's effective virtual address is within the call limiter. Typically, the segment locations defined by the call limiter are a series of jump instructions to actual entry points within the segment. The CALL instruction places the ring of the calling procedure into a program visible register. The RETURN instruction is used to transfer control back (outward) to the calling procedure; the SPM insures that the return is outward.

The transfer of data across ring boundaries is convenient because the inner ring procedure has access capabilities equal to or greater than the outer ring procedure. There are two argument validation mechanism.

The first is the calculation of Reff, the effective ring of execution, during the development of an effective virtual address. For simple memory references without indirection, Reff is equal to the current ring of execution (Rcur) as maintained by the SPM. However, in the case involving multiple indirections through many segments, the SPM will maintain in Reff the maximum value of the ring number R1 in all descriptors encountered during the preparation of the effective virtual address. Reff is set to Rcur at the beginning of each instruction cycle; for each descriptor encountered between instruction fetch and operand fetch, Reff is recomputed as the maximum of current Reff and R1 of the descriptor. The new Reff applies to fetches of all subsequent indirect addresses or data.

A second validation mechanism is provided for the more general problem where an argument pointer (i.e., a non-effective address) is copied from the outer ring to the inner ring, for instance to prevent tampering. The above mechanism will not work here, since the address constant (indirect address) now resides within the inner ring. The SPM provides a mechanism whereby an inner ring procedure can force the validation of a reference to an arbitrary virtual address with respect to any higher ring number. This can be accomplished by storing a ring number with the pointer when it is copied into the inner ring; the value of the ring number is the value of Reff computed as if the pointer were referenced directly.

On subsequent indirect references by the inner ring through this copied pointer, the SPM uses this ring number as another factor in maximizing Reff. Note again that this type of argument validation mechanism can be used only if access violation faults can be tolerated within ring 0. (See LAST in 6/43 manual.)

The SPM also provides automatic stack location upon inner ring calls. When a multi-ring gate procedure segment (R1 R2 R3) is called, it can locate the address of the stack segment for the current ring by using the value in Rcur to index into a table of preset stack pointers, since stack segment numbers are keyed by convention to the current ring number.

Process Control

The SPM implements a robust multiprogramming environment for the support of multiple processes.

Process switching time within the SCOMP is virtually the same as for the Level 6/Model 43, since the SPM adds only the overhead of loading the DBR and Rcur registers. Two of the program visible registers, status (S) and program counter (P), are automatically saved and restored upon interrupt. The remaining registers have their context stored and restored under firmware control according to a 32-bit mask. The mask, settable under program control, can be used by the save context (SAV2) and restore context (RES2) instructions to save/restore any subset of the program visible registers.

Hardware support for interprocess signalling/communication is lacking.

Summary

The evaluation of the SCOMP is summarized in Table 8.

Support for a security kernel implementation was a fundamental design goal of the SCOMP, so it is no surprise that it scored so well. What is important to note about the SCOMP is that a commerical minicomputer, the Level 6/Model 43, which provides none of the essential and convenient features, was enhanced by the SPM to provide just about all of the desired features. The SPM is designed to simply plug into the 6/43's MEGABUS in place of the commercial memory management unit. The SPM approach to providing security kernel hardware features should be applicable to other similarly bus-structured architectures.

Table 8. Honeywell - SCOMP

FUNCTIONAL AREA	ESSENTIAL FEATURES	RATING	CONVENIENCE FEATURES	RATING
Virtual Memory	Paged or Segmented Virtual Memory	*	Read-only Access	*
			Execute-only Access	+
	Null, Read-Execute Total Access Rights	*	Referenced Flag	*
			Modified Flag	*
I/O Access Control	Access to I/O devices is Controlled	*	User I/O	*
			DMA Device Access to Main Memory is Controlled	*
Execution Domains	Two Hierarchically Structured Domains or Rings	*	Three or more Domains/Rings	*
			Hierarchically Structured	*
	Controlled Transfer into Privileged Domain	*	Multiple Entry Points	*
			Argument Validation	+
			Stack Support on Domain/Ring Crossing	*
Process Control	Multiple Processes	*	Efficient Process Switch	*
			Support for Inter-Process Synchronization/Communication	-

RATING

*	Excellent
+	Good
-	Poor

DIGITAL EQUIPMENT CORPORATION

This evaluation deals only with the PDP-11/45 minicomputer offered by Digital Equipment Corporation, Maynard, Massachusetts [28]. The PDP-11/45 is at the high performance end of the PDP-11 family of upward compatible computers. The 11/45 was designed for high-speed real-time applications and, when configured with the Memory Management Unit (MMU), medium-to-large-scale, multiple user, interactive applications.

Virtual Memory

The MMU provides an unpaged segmented virtual memory, where segments may range in size from 32K words to 4,096K words in blocks of 32 words. Physical memory has a capacity of 124K words.

The logical per process address space consists of 48 4K segments. The 11/45 is a three domain machine: user, supervisor, and kernel domains. Each domain is associated with a set of 16 segment descriptor registers, so a process virtual address space consists of as many as: 16 segments in user space, 16 in supervisor space, and 16 in kernel space.

Access control information within segment descriptors define three modes of access: null, read-execute, and read-write-execute. In addition, since program and data segments can be mapped separately - in each domain, there are 8 instruction (I) space segmentation registers and 8 data (D) space segmentation registers - execute only and read only execute access modes are supported.

Referenced and modified flags are both supported within segment descriptor registers.

I/O Access Control

There are no privileged I/O instructions. Rather, control, status, and data registers for I/O devices are located in the high-order 4K words of physical memory, and I/O device registers are accessed like any other memory locations. A security kernel would restrict I/O device accesses to itself by permitting only kernel mode data segments to map into the device registers in physical memory.

Alternatively, the kernel can permit users to perform I/O by mapping user segments to requested I/O device registers. An implementation problem here is that segments must be at least 32 words long and the total I/O space is only 4K words.

Further, since DMA device accesses to main memory are not routed through the MMU, any form of user I/O must be limited to slow-speed, programmed I/O devices.

Execution Domains

As stated previously, with the MMU there are three execution domains: user, supervisor, and kernel. Which of the three sets of segmentation registers is used for address translation is determined by bits 14 and 15 of the processor status word (PSW). The three domains are hierarchically structured in terms of privilege.

Data is transferred between domains by four instructions: Move From Previous Instruction space (MFPI), Move From Previous Data space (MFPD), Move to Previous Instruction space (MTPI), and Move to Previous Data space (MTPD). MFPI, MFPD, MTPD, and MTPI are designed so that the innermost domain controls the data transfer. These instructions can be used for argument validation only if MMU faults can be tolerated within the kernel, otherwise validation must be done in software.

There are only three privileged instructions which may be executed in kernel domain only: HALT, RESET (External Bus), and Set Priority Level (in PSW). The MMU segmentation registers and the PSW are located in the high-order 4K words of physical memory - along with the I/O device registers - so the kernel domain can restrict access to these locations via memory management.

Transfer inward to kernel domain occurs as the result of all interrupts and via a set of trap instructions (EMT, TRAP, BPT, IOT). Since all trap and interrupt vectors are located in kernel virtual space, all traps and interrupts must pass through kernel space to get the new program counter (PC) and PSW. Thus, control cannot pass directly from user to supervisor space, but must be routed through kernel space. Control is passed outward by the Return from Interrupt (RTI) and Return from Trap (RTT) instructions.

Stacks are maintained for each domain; there is a stack pointer (SP) register for each domain. When a trap or interrupt occurs, the PC and PSW are pushed onto the stack pointed to by the SP specified by bits 14, 15 of the new PSW fetched from the interrupt or trap vector stored in kernel space. The old stack is restored upon RTI or RTT.

Process Control

The MMU provides the multiprogramming environment to support multiple processes.

Process switching is perhaps most inefficient on the PDP-11/45 compared to the other machines surveyed. There are two sets of 6 general purpose registers (R0 through R5), three SP registers, and one PC. Typically, the PC, the 3 SP, and one set of 6 general purpose registers must be saved/restored on a process switch. Each register must be saved/restored individually. Also, at least 32 MMU segmentation registers - minimally the 16 user and 16 supervisor - must be saved/restored, each individually.

There is no hardware support for interprocess communication.

Summary

The evaluation of the PDP-11/45 is summarized in Table 9.

Despite its relatively poor support for multiple processes, the 11/45 is rated a good candidate for an effective security kernel implementation. In fact, a prototype security kernel has already been successfully implemented and demonstrated on an 11/45 [29]. Also, security kernels to provide a secure base for a prototype secure UNIX operating system are being implemented at both MITRE and UCLA [30].

A rating of good can also be extended to the highest performance model of the PDP-11 family, the 11/70, on which the MMU is a standard feature.

DATA GENERAL ECLIPSE

The ECLIPSE line of computers offered by Data General Corporation, Southboro, Massachusetts, includes three machines, the S/100, S/200, and C/300 [32]. The latter two will be evaluated as only they may be configured with the Memory Allocation and Protection (MAP) feature for application as general purpose computer utilities.

Virtual Memory

With the MAP feature, main memory can be expanded from its standard limit of 64K bytes to a maximum of 256K bytes, in increments of 16K-byte modules. MAP provides a paged virtual memory

Table 9. PDP-11/45

FUNCTIONAL AREA	ESSENTIAL FEATURES	RATING	CONVENIENCE FEATURES	RATING
Virtual Memory	Paged or Segmented Virtual Memory	*	Read-only Access	*
			Execute-only Access	*
	Null, Read-Execute Total Access Rights	*	Referenced Flag	*
			Modified Flag	*
I/O Access Control	Access to I/O devices is Controlled	*	User I/O	+
			DMA Device Access to Main Memory is Controlled	-
Execution Domains	Two Hierarchically Structured Domains or Rings	*	Three or more Domains/Rings	*
			Hierarchically Structured	*
	Controlled Transfer into Privileged Domain	*	Multiple Entry Points	+
			Argument Validation	+
			Stack Support on Domain/Ring Crossing	*
Process Control	Multiple Processes	*	Efficient Process Switch	-
			Support for Inter-Process Synchronization/Communication	-

RATING

*	Excellent
+	Good
-	Poor

organization with a 2K byte page size. There are 3 sets of hardware mapping registers, two user maps and one map for DMA device transfers. Each user map consists of 32 mapping registers, so the logical address space per user process consists of 32 2K pages.

A single access control bit, the write-protection bit, is provided within each page descriptor; the permitted modes of per page access are null, read, and read-write. Referenced and modified bits are not supported.

I/O Access Control

The ECLIPSE computers are attractive for their support of I/O access controls. I/O instructions are not privileged. Rather, the MAP feature includes two 64-bit maps that control accesses by the two user processes to the 64 I/O devices that may be attached to the I/O bus. The active bit map permitting, the active user process may perform I/O on both slow speed and DMA I/O devices. If the bit corresponding to a given device within the active bit map is "1", the active process can execute I/O instructions on that device. If "0", the active process may not access the device and a protection fault will occur if access is attempted. This I/O device access protection can be disabled when the CPU is operating in privileged supervisor mode.

Also, as stated above, the MAP feature provides a set of mapping registers used to translate memory addresses presented by the data channel. All DMA I/O devices access main memory through the data channel, so all DMA device accesses to main memory are mapped. This mapping provides write protection on pages within the data channel's logical address space. An attempt to write (data channel input) into a write protected memory location does not cause a protection fault; rather, the attempt simply fails and a bit is set within a MAP status register.

I/O on an ECLIPSE secured by a security kernel would be controlled in the following manner. Users would request access to I/O devices from the kernel. The kernel would grant or deny access based on the security levels of the device and requesting process, the mode of access, and whether or not the device is in use. If access is granted, the kernel, operating in supervisor mode, would set the appropriate bit of the 64-bit I/O device map for the active (requesting) process. When processes are switched, an I/O device bit map must be loaded from memory with a map image for the new active process. Further, so that DMA device transfers can occur on behalf of non-active processes, the kernel must keep the data channel map consistent with the access requirements of all processes that have initiated DMA transfers.

Execution Domains

MAP provides 3 domains: two user mapped domains and one supervisor unmapped domain. There are only two levels of privilege however, user and supervisor. Supervisor domain can enable/disable MAP's various memory protection features and can access directly the first 62K bytes of main memory.

Transfer into supervisor domain occurs as the result of all interrupts and memory protection faults, and by the execution of the System Call (SYC) instruction in user domain. SYC transfers execution to a point defined by a single location in low main memory (location 2).

Data transfer between user and supervisor domain is quite convenient; there are two means of transfer at the disposal of the supervisor domain. One is use of the MAP SINGLE CYCLE instruction which permits the supervisor to use the last user map enabled for the next memory reference. The MAP SINGLE CYCLE instruction is one of several I/O instructions used to program the MAP. The MAP is treated like any other I/O device and the I/O device bit map is used to make the MAP accessible only to the supervisor domain and inaccessible to user processes. Hence use of MAP SINGLE CYCLE is restricted to supervisor domain.

The other means of data transfer is through a special mapping register for logical page 31 of supervisor address space. As noted, logical pages 0 through 30 are unmapped, meaning pages 0 through 30 of physical memory are accessed directly in supervisor domain. This special map allows the supervisor domain to transfer multiple data words to/from user space without resorting to MAP SINGLE CYCLE, which can be time consuming.

Both means of data transfer can be used to validate the data transferred, provided, of course, memory protection faults can be tolerated within supervisor domain.

No support for stack switching on domain crossing is provided.

Process Control

Addition of the MAP option provides the relocation and protection features necessary for a multiple process environment.

Although there are only two user maps, process switching is reasonably efficient because of the fast and flexible manner in which all of the maps - two user maps, the DMA map, and the two user I/O device bit maps; 112 registers in all - can be stored/loaded. Any number of contiguous registers can be stored/loaded into/from main memory by DMA at approximately 1 microsecond per register.

It takes only two instructions to save context - 4 general purpose registers, 4 stack registers, program counter - of a user process; context is saved automatically upon interrupts and faults. Only a single instruction is necessary to load a context. Total context switch latency is on the order of 15 microseconds.

There is no hardware support for interprocess communication.

Summary

The evaluation of the ECLIPSE S/200 and ECLIPSE C/300 is summarized in Table 10.

The two machines are rated fair to good candidates for an effective kernel implementation. Although relatively weak in the areas of process support, protection domains, and access rights, the ECLIPSE line offers excellent hardware support in the area of I/O control.

HEWLETT PACKARD

Only one Hewlett Packard computer system, the HP 3000 Series II [32], is evaluated. The HP3000/II is a general purpose computer utility designed for both batch and interactive data processing. It is a stack based architecture and as such provides many powerful operating features; e.g., shared, reentrant, and recursive code segments, efficient parameter passing and subprogram linkage. Three models are available; 5, 7, and 9, and this evaluation applies to all three.

Virtual Memory

The HP3000 provides a segmented virtual memory organization which is tailored toward its stack based architecture. Code segments may be 32K bytes in length; data segments may range up to 64K bytes. Physical main memory ranges from 128K bytes of semiconductor memory up to 512K bytes, in modules of 64K bytes.

Table 10. Data General - ECLIPSE

FUNCTION AREA	ESSENTIAL FEATURES	RATING	CONVENIENCE FEATURES	RATING
Virtual Memory	Paged or Segmented Virtual Memory	*	Read-only Access	-
			Execute-only Access	-
	Null, Read-Execute Total Access Rights	*	Referenced Flag	-
			Modified Flag	-
I/O Access Control	Access to I/O devices is Controlled	*	User I/O	*
			DMA Device Access to Main Memory is Controlled	*
Execution Domains	Two Hierarchically Structured Domains or Rings	*	Three or more Domains/Rings	*
			Hierarchically Structured	-
	Controlled Transfer into Privileged Domain	*	Multiple Entry Points	-
			Argument Validation	+
			Stack Support on Domain/Ring Crossing	-
Process Control	Multiple Processes	*	Efficient Process Switch	+
			Support for Inter-Process Synchronization/Communication	-

RATING

* Excellent
+ Good
- Poor

Program addressing is not descriptor based in the conventional sense, however, and access checking capabilities are not supported - a serious deficiency. In all of the virtual or mapped memory systems examined thus far, pages or segments are defined by and addressed through descriptors. User program addresses are virtual and are translated into physical main memory addresses using descriptor information, and process access rights to the accessed page or segment are checked. On the HP3000, all code and data segments currently in use on the system are defined by four-word entries within a global code segment table (CST) and global data segment table (DST). These entries contain some of the information that conventional segment descriptors usually hold: an absence bit, referenced bit, segment address in main memory (if resident) or on disk (if not resident); but since these entries are global, they do not contain access control information which is conventionally process local.

Program addressing works in the following manner. A set of CPU registers defines the current executing code segment: the start and end of the segment and the current point of execution (program counter). Another set of registers defines the current data segment, some portion of which is treated as a stack; these registers define the start and end of the data segment and the beginning and top of the stack.

All instruction fetches are from the current code segment. Transfer of execution to another code segment is accomplished by a special hardware instruction (PCAL), which uses linkage information contained in a segment transfer table (STT) within the current code segment to locate the new code segment entry in the CST. PCAL may fault to privileged software if the addressed code segment contains information which PCAL uses to determine whether the transfer is legal.

All operands are fetched/stored from/into the current data segment. A process has both read and write access to its current data segment, since it is not possible to grant a process just read access or write access. The process must invoke the supervisor to change its current data segment.

I/O Access Control

I/O instructions can only be executed in privileged processor mode; user processes cannot do I/O.

The selector channel accesses main memory directly using absolute memory addresses; DMA devices accesses to memory are not mediated. Note also that programmed I/O through the multiplexer channel involves unmediated direct access to main memory.

Execution Domains

Two execution domains are provided: privileged and user domain. Software executing in privileged domain can 1) execute privileged instructions, 2) directly address all areas of physical memory, and 3) invoke code segments that have been declared uncallable (i.e., the uncallable bit is set in the local program label or STT entry). Also, instruction fetches to the current code segment and data references to the current data segment are not subject to bounds checking. Stack underflow is also permitted in privileged domain.

A security kernel must run alone in privileged domain. An operating system must run as a process in user domain.

User software initiated transfer into privileged domain is accomplished by the Procedure Call (PCAL) instruction. PCAL uses linkage information resident within the calling and called code segments' segment transfer tables (STT). Entrance into privileged domain results from the invocation of a procedure contained within a code segment assigned to privilege domain. PCAL is very flexible because multiple entry points into privileged domain segments are provided through STT linkage information.

Parameters are passed to privileged domain software on the user's current data stack. Pointers that are displacements from the top of the users stack may be passed as parameters. Validation that these pointers reference locations within the bounds of the user's stack must be performed entirely by software.

Separate stacks are maintained automatically in user and privileged domain. On transfers into privileged domain via external and most internal interrupts, an interrupt control stack (ICS) is set up by the hardware implemented interrupt handler.

Process Control

The segmented virtual memory organization provides a good environment for multiple processes, if the environment is properly managed by privileged domain software. Hardware support for a fast process switch is provided.

There are a number of processor registers that are automatically saved on interrupts by the hardware interrupt handler:

current code and data segment registers, index register, status register, top of stack registers. The contents of these registers are saved on the user process' stack. Two instructions (SETR, PSHR) are provided to save/load on/from the current stack any subset of the above registers. Note, to load (S-bank, DB, DL, Z, status) or store (DB, DB-bank, S-Bank) some of these registers require privileged domain operation. Unlike a lot of virtual memory machines, no mapping registers need be saved/loaded during a process switch. Just memory locations 1 (pointer to CST extension for current program) and 4 (pointer to process control block for current process) must be loaded to define the address space of the new process.

There is no support for interprocess communication or synchronization.

Summary

The evaluation is summarized in Table 11. Because the essential access checking capability on data segment access is not provided, the HP3000 Series II is rated an extremely poor choice for an effective kernel implementation.

Table 11. Hewlett-Packard 3000 - Series II

FUNCTIONAL AREA	ESSENTIAL FEATURES	RATING	CONVENIENCE FEATURES	RATING
Virtual Memory	Page or Segmented Virtual Memory	*	Read-only Access	-
			Execute-only Access	*
	Null, Read-Execute Total Access Rights	-	Referenced Flag	*
			Modified Flag	-
I/O Access Control	Access to I/O devices is Controlled	*	User I/O	-
			DMA Device Access to Main Memory is Controlled	-
Execution Domains	Two Hierarchically Structured Domains or Rings	*	Three or more Domains/Rings	-
			Hierarchically Structured	-
	Controlled Transfer into Privileged Domain	*	Multiple Entry Points	*
			Argument Validation	-
			Stack Support on Domain/Ring Crossing	+
Process Control	Multiple Processes	*	Efficient Process Switch	*
			Support for Inter-Process Synchronization/Communication	-

RATING

*	Excellent
+	Good
-	Poor

SECTION IV

CONCLUSION

Table 12 is an attempt to rate the various machines with respect to each other. To do so, a rating from 0 to 4 is applied to each vendor's machine, or series of machines, in each of the four areas of evaluation. A rating of 0 is assigned if the essential features are not provided; a rating of 1 applies if the essentials are provided, but none of the conveniences are; a rating of 4 applies if all essentials and conveniences are provided; ratings between 1 and 4 are assigned depending on the number of conveniences.

The SCOMP and the PRIME machines are clearly the best candidates. SCOMP was designed to support a security kernel and rates the best in the area of I/O control; the SCOMP's Security Protection Module (SPM) is a descriptor-based general access controller that includes I/O devices within the virtual environment. PRIME rates the best on Process Control because of its innovative support for interprocess synchronization and its shared segment table arrangement.

The PDP-11/45 stands next, all alone; the 11/45 has already exhibited itself as a good hardware base for a kernel implementation.

The ECLIPSE line, GA-16/440, Varian 70 Series, MODCOMP IV/35, IBM Series 1/Model 5, and INTERDATA 8/32 all rate roughly the same; all provide the essentials and kernel implementations are certainly feasible. MODCOMP IV/35 is notable for its strong support for multiprogramming - as many as 15 sets of mapping registers for user processes. The ECLIPSE line rates highly for its I/O protection. INTERDATA's Load Real Address (LRA) instruction is the best solution of all for argument validation. LRA converts a user virtual address to a physical address and sets condition codes on an access violation rather than generating a fault, which may not always be tolerable in privileged domain.

The HP3000 Series II rates as a bad architectural base because of its lack of access checking capability.

Table 12. Summary

	VIRTUAL MEMORY	I/O ACCESS CONTROL	EXECUTION DOMAINS	PROCESS CONTROL	TOTAL
PRIME	4	2	4	4	14
SCOMP	4	4	4	2	14
PDP-11/45	4	2	3	1	10
ECLIPSE	1	4	2	1	8
GA-16/440	3	2	2	1	8
V70 Series	3	1	2	1	7
MODCOMP	2	1	2	2	7
IBM 1/5	2	1	2	1	6
INTER-DATA 8/32	2	1	1	1	5
HP-3000/II	0	1	2	2	5

RATING

0	=	Essentials not provided
1	=	Essentials prcvided/no conveniences
4	=	Essentials provided/all conveniences
2,3	=	Depending on conveniences provided

REFERENCES

1. L. Smith, "Architectures for Secure Computing Systems," ESD-TR-75-51, Electronic Systems Division, AFSC, Hanscom AFB, MA, April 1975 (ADA009221).

2. "ESD 1974 Computer Security Developments Summary," MCI-75-1, Electronics Systems Division (AFSC), L.G. Hanscom AFB, Bedford, Massachusetts, December 1974.

3. J.P. Anderson, "Computer Security Technology Planning Study," ESD-TR-73-51, Volumes I and II, James P. Anderson Co., Fort Washington, Pennsylvania, October 1972.

4. R.W. Conway, et al., "On the Implementation of Security Measures in Information Systems," Communications of the ACM, April 1972.

5. K.G. Walter, et al., "Primitive Models for Computer Security," ESD-TR-74-117, Case Western Reserve University, Cleveland, Ohio, January 1974.

6. D.E. Bell and L.J. LaPadula, "Secure Computer Systems," ESD-TR-73-278, Volumes I-III, Electronic Systems Division, AFSC, Hanscom AFB, MA, November 1973 - June 1974 (AD770768, AD771543, AD780528).

7. D.E. Bell and L.J. LaPadula, "Secure Computer System: Unified Exposition and Multics Interpretation," ESD-TR-75-306, Electronic Systems Division, AFSC, Hanscom AFB, MA, March 1976 (ADA023588).

8. D. Branstad, "Privacy and Protection in Operating Systems," Computer, Volume 6, Number 1, January 1973, pp. 43-47.

9. K.J. Biba, "Integrity Considerations for Secure Computer Systems," ESD-TR-76-372, Electronic Systems Division, AFSC, Hanscom AFB, MA, April 1977 (ADA039324).

10. T.A. Linden, "Operating System Structures to Support Security and Reliable Software," ACM Computing Surveys, Volume 8, Number 4, December 1976, pp. 409-445.

11. M.D. Schroeder and J.M. Saltzer, "A Hardware Architecture for Implementing Protection Rings," Communications of the ACM, Volume 15, Number 3, March 1972, pp. 157-170.

12. D.E. Bell and E.L. Burke, "A Software Validation Technique for Certification: The Methodology," ESD-TR-75-54, Electronic Systems Division, AFSC, Hanscom AFB, MA, April 1975 (ADA009849).

13. D.L. Parnas, "A Technique for Software Module Specification With Examples," Communications of the ACM, Volume 15, Number 5, May 1972, pp. 330-336.

14. W.R. Price, "Implications of a Virtual Memory Mechanism for Implementing Protection in a Family of Operating Systems," Ph.D. Thesis, Carnegie-Mellon, at the University, Pittsburgh, Pennsylvania, June 1973.

15. J.K. Millen, "Security Kernel Validation in Practice," Communications of the ACM, Volume 19, May 1976, pp. 243-250.

16. D.K. Kallman and J.K. Millen, "Security Kernel Verification Techniques: Algorithmic Representation," ESD-TR-78-123, Electronic Systems Division, AFSC, Hanscom AFB, MA, April 1978 (ADA054098).

17. L. Robinson, P.G. Neumann, K.N. Levitt, and A.R. Saxena, "On Attaining Reliable Software for a Secure Operating System," 1975 International Conference on Reliable Software, Los Angeles, California, April 1975, pp. 267-284.

18. E.L. Burke, "Secure Minicomputer Architectures," M76-224, The MITRE Corporation, Bedford, Massachusetts, October 1976.

19. Modular Computer System Inc., "MODCOMP IV Central Processor," Form 210-110000-000, May 1975.

20. General Automation, Inc., "GA-16/440 System Reference," Form 88A00466, September 1975.

21. International Business Machines Corporation, "Series/1 Model 5 4955 Processor and Processor Features Description," Form GA34-0021-0, November 1976.

22. Varian Data Machines, "Varian 70 Series Architecture Reference Manual," Form 98A99061000, August 1976.

23. Varian Data Machines, "Varian 70 Series Memory Map Manual," Form 98A9906101, September 1975.

24. Interdata, Inc., "Model 8/32 Processor User's Manual," Form 29-428R01, 1975.

25. W.L. Schiller, "The Design and Abstract Specification of a Multics Security Kernel," ESD-TR-77-259, Vol. I, Electronic Systems Division, AFSC, Hanscom AFB, MA November 1977 (ADA048576).

26. M. Gasser, "Top Level Specification of a Security Kernel for Multics Front-End Processor," ESD-TR-77-258, Electronic Systems Division, AFSC, Hanscom AFB, MA, November 1977 (ADA047309).

27. Honeywell Information Systems, Inc., "Honeywell Level 6 Minicomputer Handbook," Form AS22, Revision 0, January 1976.

28. Digital Equipment Corporation, "PDP-11/45 Processor Handbook," 1973.

29. W.L. Schiller, "The Design and Specification of a Security Kernel for the PDP-11/45," ESD-TR-75-69, Electronic Systems Division, AFSC, Hanscom AFB, MA, May 1975 (ADA011712).

30. M. Kampe, C.C. Kline, G.J. Popek, and E. Walton, "The UCLA Data Secure UNIX Operating System," The University of California at Los Angeles, Los Angeles, California, (draft) September 1976.

31. Data General Corporation, "ECLIPSE LINE COMPUTERS: Programmer's Reference Manual," Form 015-000024-04, March 1975.

32. Hewlett-Packard Company, "HP3000 Series II Computer System: System Reference Manual," Form 30000-90020, June 1976.